FlashPro-LM

Flash Programmer for Texas Instruments' Stellaris ARM Family

User's Manual

Software version 1.0

Elprotronic Inc.

Elprotronic Inc.

16 Crossroads Drive
Richmond Hill,
Ontario, L4E-5C9
CANADA

Web site: www.elprotronic.com
E-mail: info@elprotronic.com
Fax: 905-780-2414
Voice: 905-780-5789

END USER LICENSE AGREEMENT

PLEASE READ THIS DOCUMENT CAREFULLY BEFORE USING THE SOFTWARE AND THE ASSOCIATED HARDWARE. ELPROTRONIC INC. AND/OR ITS SUBSIDIARIES ("EL-PROTRONIC") IS WILLING TO LICENSE THE SOFTWARE TO YOU AS AN INDIVIDUAL, THE COMPANY, OR LEGAL ENTITY THAT WILL BE USING THE SOFTWARE (REFER-ENCED BELOW AS "YOU" OR "YOUR") ONLY ON THE CONDITION THAT YOU AGREE TO ALL TERMS OF THIS LICENSE AGREEMENT. THIS IS A LEGAL AND ENFORCA-BLE CONTRACT BETWEEN YOU AND ELPROTRONIC. BY OPENING THIS PACKAGE, BREAKING THE SEAL, CLICKING I AGREE BUTTON OR OTHERWISE INDICATING AS-SENT ELECTRONICALLY, OR LOADING THE SOFTWARE YOU AGREE TO THE TERMS AND CONDITIONS OF THIS AGREEMENT. IF YOU DO NOT AGREE TO THESE TERMS AND CONDITIONS, CLICK ON THE I DO NOT AGREE BUTTON OR OTHERWISE INDI-CATE REFUSAL, MAKE NO FURTHER USE OF THE FULL PRODUCT AND RETURN IT WITH THE PROOF OF PURCHASE TO THE DEALER FROM WHOM IT WAS ACQUIRED WITHIN THIRTY (30) DAYS OF PURCHASE AND YOUR MONEY WILL BE REFUNDED.

1. License.
The software, firmware and related documentation (collectively the "Product") is the property of Elprotronic or its licensors and is protected by copyright law. While Elprotronic continues to own the Product, You will have certain rights to use the Product after Your acceptance of this license. This license governs any releases, revisions, or enhancements to the Product that Elprotronic may furnish to You. Your rights and obligations with respect to the use of this Product are as follows:

YOU MAY:

- A. use this Product on many computers;

- B. make one copy of the software for archival purposes, or copy the software onto the hard disk of Your computer and retain the original for archival purposes;

- C. use the software on a network

YOU MAY NOT:

- A. sublicense, reverse engineer, decompile, disassemble, modify, translate, make any attempt to discover the Source Code of the Product; or create derivative works from the Product;

- B. redistribute, in whole or in part, any part of the software component of this Product;

- C. use this software with a programming adapter (hardware) that is not a product of Elpro-tronic Inc.

2. Copyright

All rights, title, and copyrights in and to the Product and any copies of the Product are owned by Elprotronic. The Product is protected by copyright laws and international treaty provisions. Therefore, you must treat the Product like any other copyrighted material.

3. Limitation of liability.

In no event shall Elprotronic be liable to you for any loss of use, interruption of business, or any direct, indirect, special, incidental or consequential damages of any kind (including lost profits) regardless of the form of action whether in contract, tort (including negligence), strict product liability or otherwise, even if Elprotronic has been advised of the possibility of such damages.

4. DISCLAIMER OF WARRANTIES.

You agree that Elprotronic has made no express warranties to You regarding the software, hardware, firmware and related documentation. The software, hardware, firmware and related documentation being provided to You "AS IS" without warranty or support of any kind. Elprotronic disclaims all warranties with regard to the software and hardware, express or implied, including, without limitation, any implied warranties of fitness for a particular purpose, merchantability, merchantable quality or noninfringement of third-party rights.

FC

NOTE: This equipment has been tested and found to comply with the limits for a Class B digital devices, pursuant to Part 15 of the FCC Rules. These limits are designed to provide reasonable protection against harmful interference in a residential installation. This equipment generates, uses, and can radiate radio frequency energy and, if not installed and used in accordance with the instruction manual, may cause harmful interference to radio communications. However, there is no guarantee that interference will not occur in a particular installation. If this equipment does cause harmful interference to radio or television reception, which can be determined by turning the equipment off and on, the user is encouraged to try to correct the interference by one of more of the following measures:

- Reorient or relocate the receiving antenna,

- Increase the separation between the equipment and receiver,

- Connect the equipment into an outlet on a circuit different from that to which the receiver is connected,

- Consult the dealer or an experienced radio/TV technician for help.

Warning: Changes or modifications not expressly approved by Elprotronic Inc. could void the user's authority to operate the equipment.

C E

Contents

List of Figures

Chapter 1

Introduction

The FlashPro-LM Programmer is designed to program the LM series MCU devices from Texas Instruments. The programmer configures target devices by using the **debug** interface.

The programmer package consists of a microcontroller based USB-FPA adapter (Flash Programming Adapter, shown in Figure 1.1), Windows™ based software, cable to connect the adapter with the computer's USB port and ribbon cables with cable converter adapter. The internal firmware software allows to communicate with the programmed device at high speed. The effective programming speed (write only) is around **30 kB/s** . Due to this high speed communication, programming time is very short and programmer can be used to program flash devices in the production process. For example, a device with **128 kB FLASH** , can be programmed in **8 seconds** . This time includes initialization, erasing memory, blank checking, programming and fast verification.

To simplify the production process the programming software package can assign a serial number, model type, and revision. Each serial number is unique for each programmed device and is assigned automatically. Several serial number formats are available.

There are a number of erase and write options available as well. This enables the user to erase and write all flash memory or only a specified fragment of memory. This feature is very useful when only a part of programmed data or code needs to be replaced. For example one can change the serial number, calibration data or personality data without erasing existing program code.

Figure 1.1: The FlashPro-LM Programmer package.

Chapter 2

Features

The FlashPro-LM Programmer is designed to program the LM series MCU devices from Texas Instruments. To facilitate high speed communication an application software for the programming adapter has been optimized for maximum speed.

2.1 Key Features

The key features of the FlashPro-LM Programmer are:

- Supports all LM-M3 and LM-M4 devices from Texas Instruments.

- Programming speed via debug interface is approximately **30 kB/s** .

- Our programmers are professionally made and are **recommended by Texas Instruments** as the Third Party Tools source.

- Full memory or page memory erase capability.

- Write Check Sum verification.

- No code size limitations.

- Target device can be powered from the programming adapter or from external source.

- Easy to use Windows™ based software.

- Programmer accepts TI (*.txt), TI CCS (*.bin), Motorola (*.s19) and Intel (*.hex) data files for programming.

- Lock setup capability, useful in production.

- Software package can assign and automatically increment a **serial number** , model type and revision. Serial Number with or without an automatically inserted current date can be stored in Flash memory in HEX, BCD or ASCII format.

- Log file capability allows to review information about flashed target devices.

- **DLL** software package can control programmer from other programs.

- Driver for the IAR CSpy - EW8051 for debugging using FPA and IAR Embedded Workbench IDE software.

- Programmer has been fully tested to comply with **FCC** and **CE** requirements.

- Uses USB-1.1 (**12 Mb/s**) port to communicate with the programming adapter.

2.2 Custom Features

The FlashPro-LM Programmer can be controlled from external software as well as custom scripts to specify programming sequences. These features are very useful for automation in a production environment.

2.2.1 Encrypted Image option

Code contents downloaded to target device can be encrypted and blocked against unauthorized access.

2.2.2 Script file

A user can define a sequence of programming steps by means of a script file. The script file is a sequence of programmer commands, where each command corresponds to a button in the programming software. Each command can be accompanied by a few options. A script file of up to 1000 lines can be created. The detailed description of script commands is given in Chapter 10. Please note that the script file is not available in the **lite** version of the programming software.

2.2.3 DLLs

The programming adapter can also be controlled through user created applications. For this purpose a DLL is provided to allow a user to develop custom applications that can control the programming adapter and allow the programming of target devices via the FlashPro-LM Programmer .The Multi-FPA API-DLLs allows to fully control up to 16 programming adapters (to program simultaneously up to 16 target devices) from external software written in MS Visual C++, MS Visual Basic, LABView, DOS or other programming packages like Borland C++ etc. See the "FlashPro-LM Programmer - Remote Control Programming User's Guide" for details.

2.2.4 Driver for the IAR CSpy and KEIL uVision debugger

The FPA programming adapter can also be used with the IAR Embedded Workbench or KEIL uVision IDE software for debugging. See Chapters 11 and 12 for details.

Chapter 3

Getting Started

FlashPro-LM Programmer package contains:

1. One READ ME FIRST document.

2. One FlashPro-LM Programmer CD-ROM (Software + Manual).

3. One FlashPro-LM Programmer (USB-FPA 4.x).

4. One 6 feet long USB-A to USB-B cable.

5. FlashPro-LM Programmer that allows to connect target device to USB-FPA adapter.

6. One 10-pins ribbon cable.

7. One 14-pins ribbon cable.

The FlashPro-LM Programmer software runs under Windows™ ME, WinNT, 2000, XP, and
7. Follow instructions below to install the software:

1. Insert FlashPro-LM Programmer Software CD into your CD-ROM drive.

2. The FlashPro-LM Programmer Setup wizard appears automatically. Click **Install FlashPro-LM Programmer** to begin the installation process.

3. If the Setup wizard does not start automatically, click the Start button and choose the Run dialogue box. Execute "D:\SETUP.EXE", where D represents the drive letter of your CD-ROM drive. Then click the OK button.

4. Once the installation program starts, on-screen instructions will guide you through the remainder of the installation. You must accept the license agreement before using this software.

Figure 3.1: New Hardware Wizard, step 1. Select "Yes, this time only."

3.1   Driver Installation

The installation program will place the USB driver files in Windows directories "Windows\inf" and "Windows\system32\drivers".

1. Plug in the FlashPro-LM Programmer to the PC USB Port, using provided cable extender (USB-A to USB-B).

2. **For Windows XP, Vista, 7** . The "New hardware has been found - USB-FPA-BOOT" window should be displayed. Follow wizard instructions to install the drivers.

   (a) In the first Wizard dialogue screen (see Figure 3.1) select the "Yes, this time only" option.

   (b) In the second Wizard dialogue screen (see Figure 3.2) select the "Install the software automatically (Recommended)" option and press NEXT button.

   (c) Software will search for the USB-FPA-BOOT driver. It can take a few minutes to find the necessary files. When the driver is found then the following warning will be displayed

---
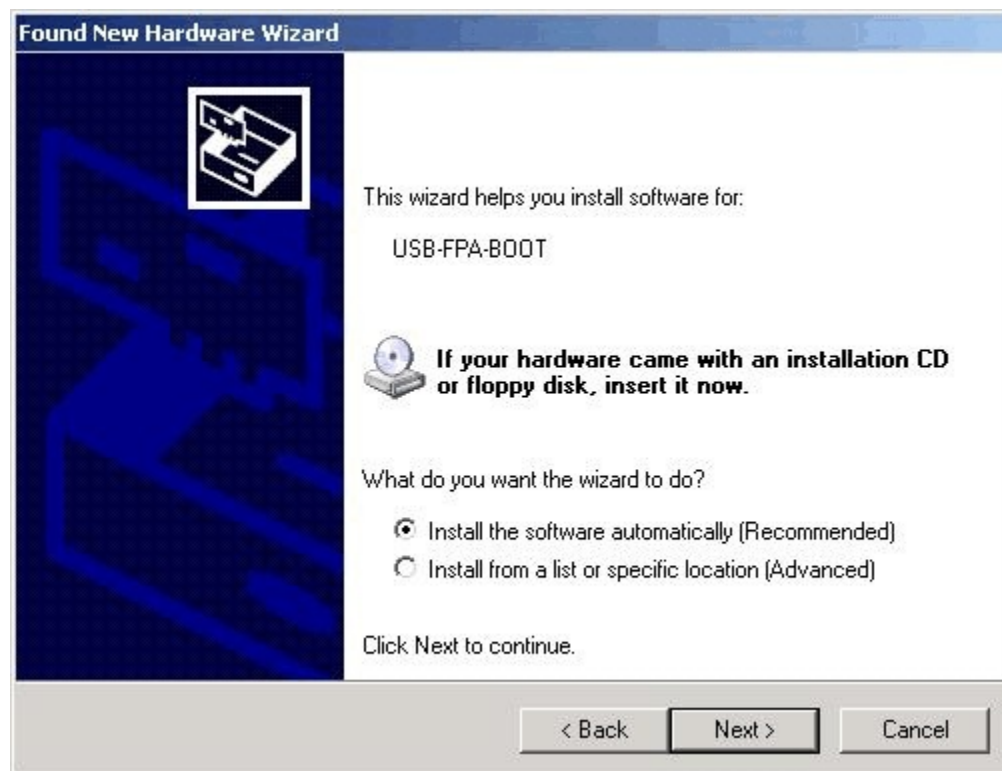
Figure 3.2: New Hardware Wizard, step 2. Select "Install the software automatically."
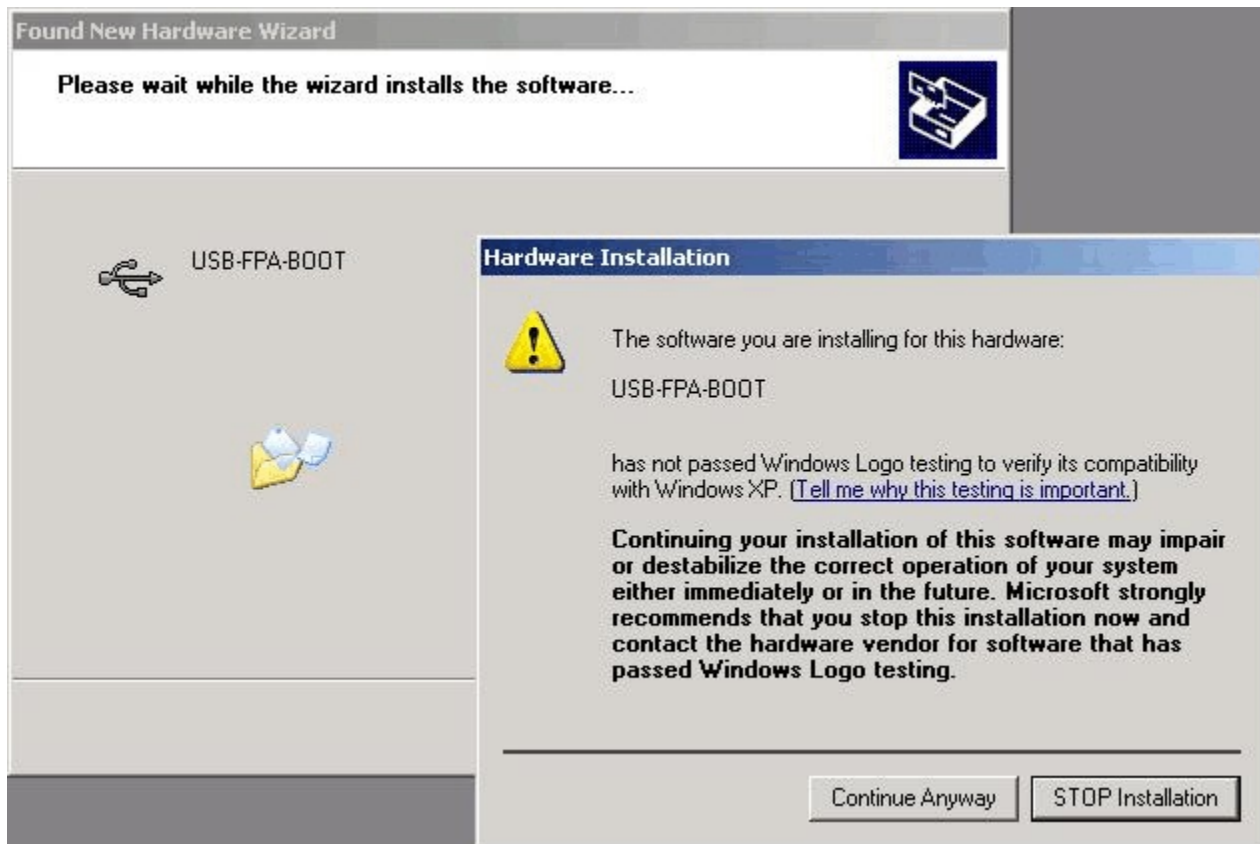
Figure 3.3: New Hardware Wizard, step 3. Ignore this warning.

Figure 3.4: New Hardware Wizard, step 4. USB-FPA-BOOT driver installed.

Figure 3.5: New Hardware Wizard, step5. USB-FPA (Elprotronic) driver installed.

(Figure 3.3).

(d) Ignore this message and press button "Continue Anyway". The first USB-FPA-BOOT driver should be installed and the following message will be displayed (Figure 3.4).

(e) Press the "Finish" button.

After a few seconds, a second "New hardware" window will appear regarding the USB-FPA (Elprotronic). Repeat procedures described above to install the USB-FPA (Elprotronic) driver (see Figure 3.5). If for any reason the wizard cannot find the USB driver's location then use the manual browse option to locate driver files in the software directory "C:\Program Files\Elprotronic\Drivers USB-FPA\XP,Vista,Win-7".

For Windows 2000, 98-SE, ME . The "New hardware has been found" window will be displayed instead. Follow wizard instructions to install the drivers.

(a) Press "Next" when the Device Wizard Driver screen appears.

(b) Select the following option on the wizard screen: **select a suitable driver for my device (recommended)** and press "Next".

(c) Select the third option - "Specify a location" for a location of the driver files.

(d) From the browser select the "D:\drivers\W2K, W98SE, WinME" directory, where "D:" is the CD-ROM drive location, or in the application software directory "C:\Program Files\Elprotronic\Drivers USB-FPA\W2K, W98SE, WinMe" and press "Next".

(e) Driver installation process will start.

3. Driver installation procedures should be done twice to install two USB drivers - the Boot driver and the Application driver.

4. Reboot computer if necessary.

3.2   Hardware Setup

1. Connect the USB-FPA Flash Programming Adapter to the PC USB Port or via USB-HUB using provided cable extender (USB-A to USB-B) (see Figure 3.6).

2. Plug in socket connector from the USB-FPA Flash Programming Adapter to the J1 connector on the FlashPro-LM Programmer Adapter (PN: PM036A1) using ribbon cable with 14-pins connectors. Connect target device to FlashPro-LM Programmer using ribbon cable with 10-pins connectors. Make sure that pin 1 on your device board's header is connected to pin 1 (red wire) of the socket connector.

The FlashPro-LM Programmer software package (GUI or Multi-FPA API-DLL) can control the FlashPro-LM Programmer (USB-FPA-4.x).
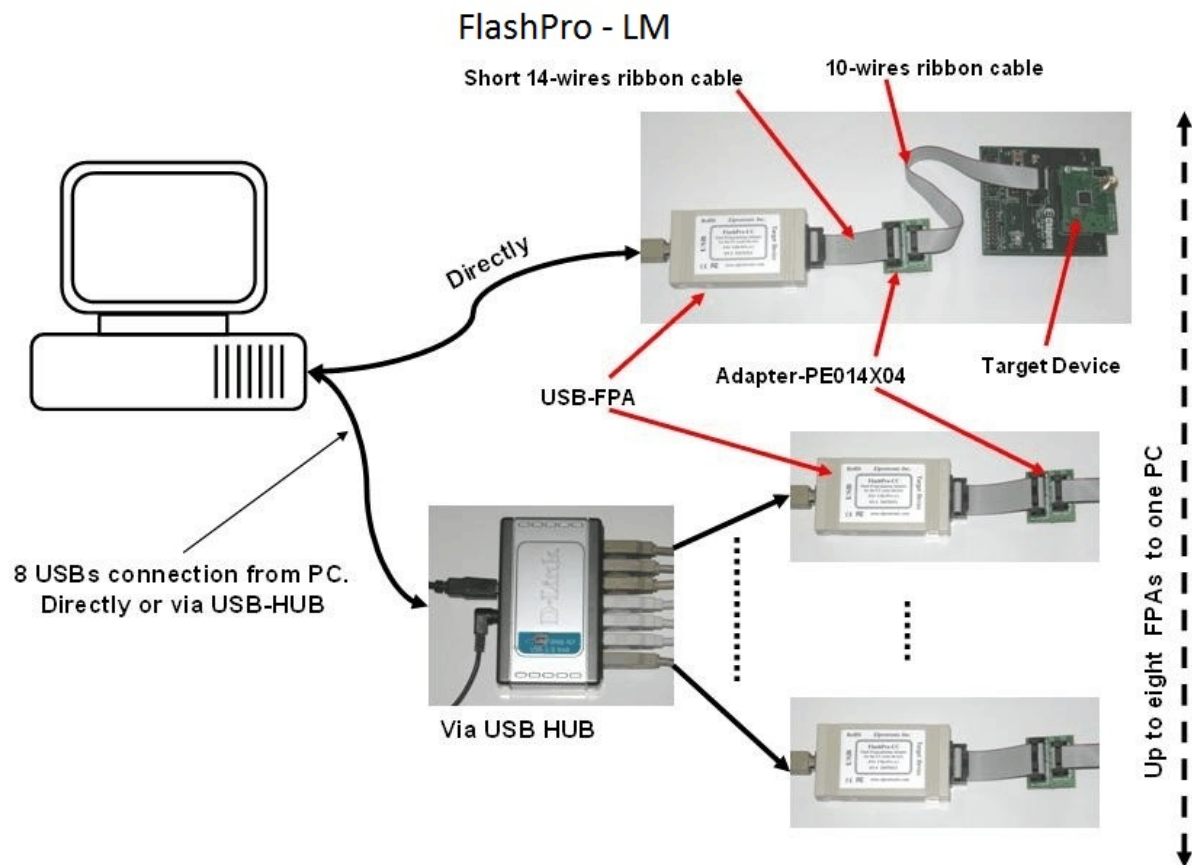
Figure 3.6: Hardware setup, cable connections.

3.3   Starting up FlashPro-LM Programmer

To start the FlashPro-LM Programmer click on the FlashPro-LM Programmer Elprotronic icon.

Once started the software will attempt to access the programming adapter. If no error messages appear then the software has initialized without a problem and you may begin using it. However, if the programming adapter is not detected an error message will appear. To correct the problem, make sure that the connection cable is properly attached and the USB driver is installed. Several warning messages regarding a mismatch between flash size and code file size may appear, but you can ignore them at this point. They are designed to warn the user before attempting to program a code file that is too large for a selected MCU device type. These warning messages are relevant after you have selected your desired MCU device type and code file, since code outside of flash will not be programmed.

3.4   FlashPro-LM Programmer Selector

The FlashPro-LM Programmer software has a Multi-USB feature. Up to 16 Flash Programming Adapters can be connected to one PC. Each adapter can be controlled by a separate instance of the programming software application. Up to 16 applications can be opened at the same time. Each application can have independent setup from the others (code file, controlled microcontroller type etc.).

When more then one FlashPro-LM Programmer is connected to a PC then each time you start the programmer application a FlashPro-LM Programmer Selector dialogue screen will appear (see Figure 3.7). The dialogue screen will list all adapters connected and allow you to choose the adapter you wish the application to control. Make sure that the selected FPA is not used by another opened application. The selected FPA's serial number will be displayed on the bottom left side of the programming dialogue screen.

When the Multi-FPA API-DLL is used, then all adapters can be controlled from one software application. See the "FlashPro-LM Programmer - Remote Control Programming User's Guide" for details.
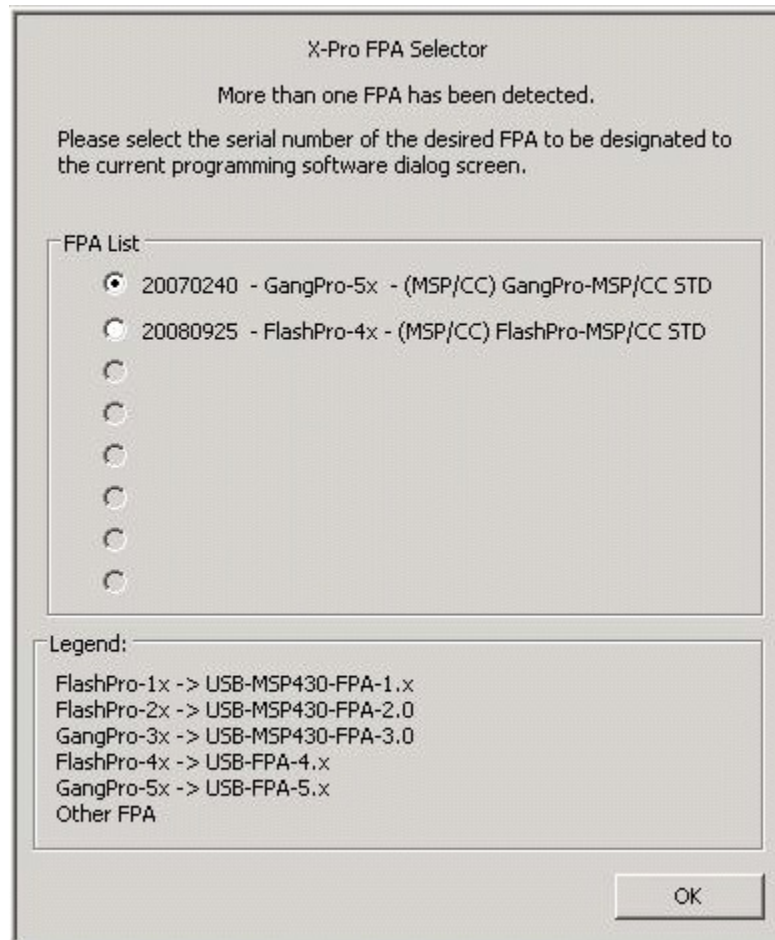
Figure 3.7: FPA selector. When one or more programmers are connected to the computer at once, the FPA selector allows the user to choose which programmer to operate.

Chapter 4

Programming Dialogue Screen

The programming dialogue box (see Figure 4.1) contains a pull down menu, interface selection box, lock protection bits box, device action buttons, report (status) window, open file buttons, target device information box, serial number box, power DC status and check sum result boxes.

All device action buttons, power ON/OFF button and the check sum result box have their own status indicators. Each indicator can assume any of the following conditions:

- ▢ - idle status.

- ⬤ - Test in progress. For power on/off - DC voltage is correct.

- ⬤ - access enabled.

- ⬤ - access denied. For power ON/OFF - DC voltage is too low (below 2.0V).

- ☑ - device action has finished successfully.

- ☒ - device action has finished, but result failed.

- ☒ - applies to blank check only. All of memory is not clean, but the specified memory segment is clean.

4.1   MCU Device Type

Target MCU device type can be selected from the pull down field of the Family, Group, and Name menus shown in Figure 4.2. The pull down fields contain a list of all devices in the CM3 and CM4 family series currently available.

When communication between the microcontroller and the programming adapter is initialized, the software will detect the target microcontroller automatically. The type of detected microcontroller is displayed in the field "Target:". This allows the software to warn the user if the connected microcontroller does not match the one specified during configuration.

15
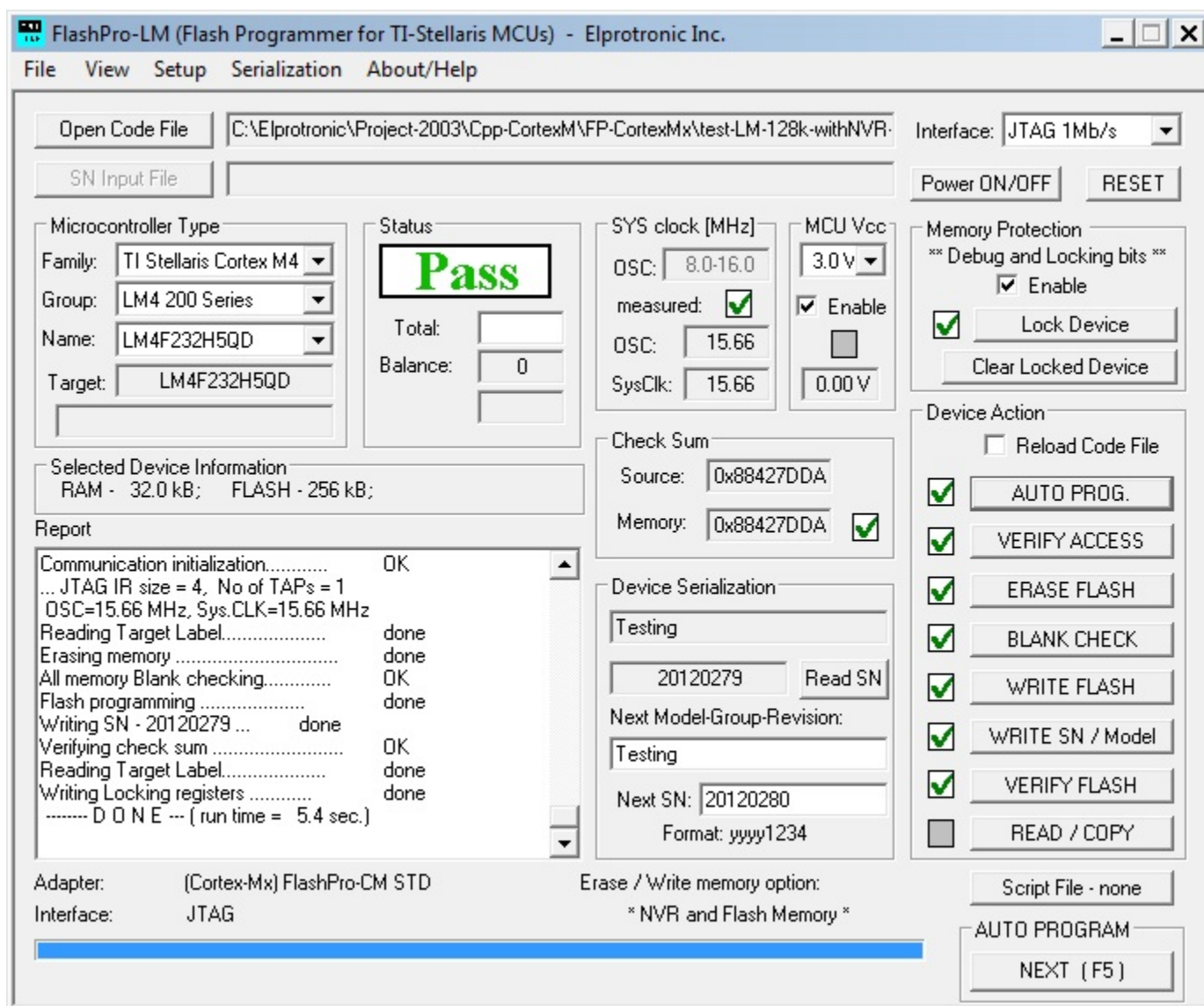
Figure 4.1: Programming Dialogue Box Screen.
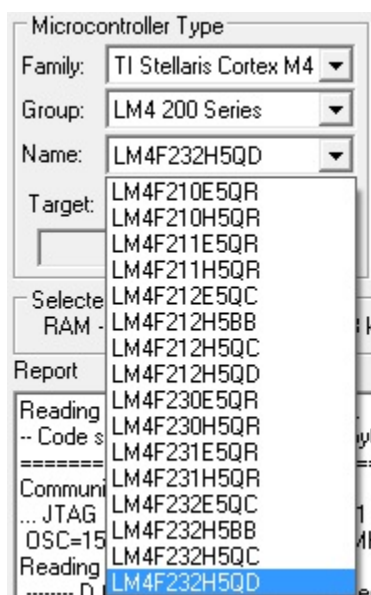
Figure 4.2: The MCU type can be selected here.

4.2   Code File Management

The FlashPro-LM Programmer provides a few options to manage code files. These options allow the user to open a code file, and save memory or read code data into a new code file. The "Open Code File" button, or the **File→Open Code File** option prompts for opening the object file that contains the code data, as shown in Figure 4.3. When the file is selected the contents of the file are downloaded into application memory. If the selected target device does not have enough flash memory to fit the data contained in the code file, a warning message as shown in Figure 4.4 will appear.

When a code file is opened and read successfully the code file name and full path will be displayed on the right side of the "Open Code File" button (see Figure 4.1). Contents of the selected file can be viewed by selecting the **View→Code File Data** option (see Chapter 5).

The **File→Save Code as...** option saves the data currently contained within the application code data block into a code file. When the user selects this option the window in Figure 4.5 will appear, prompting for the name of the file to be created. The **File→Save Memory as...** option will save the code read from a target device into a code file similarly to the **File→Save Code as...** option. All of the aforementioned Code File options work with the three most popular code file formats. These formats are the Texas Instruments, the Motorola and the Intel file formats. FlashPro-LM Programmer will work with any of these formats and will easily convert one file format to another by using the **File→Open Code File** and **File→Save Code as...** options.
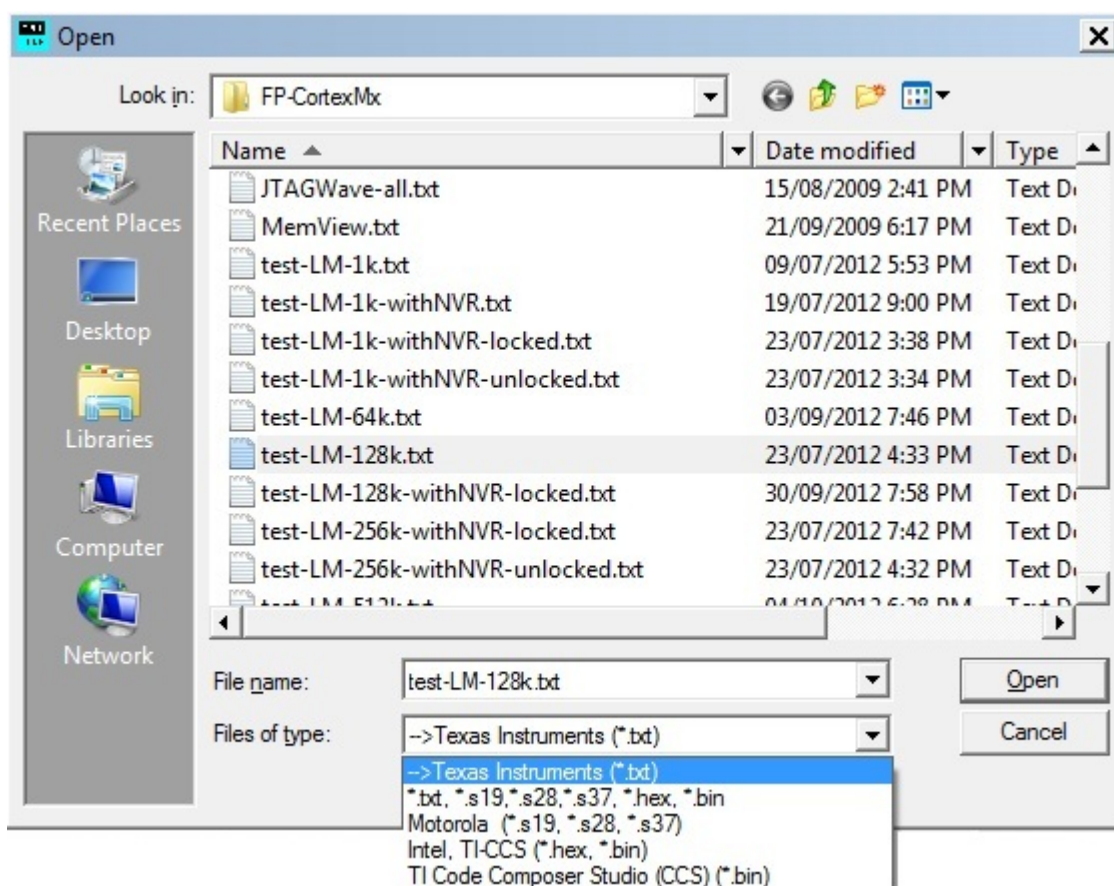
Figure 4.3: Select the Code file you wish to program to the MCU.
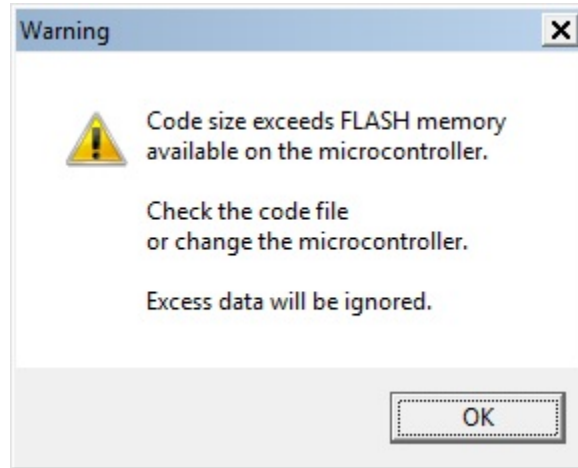
Figure 4.4: Code size exceeds Flash size.
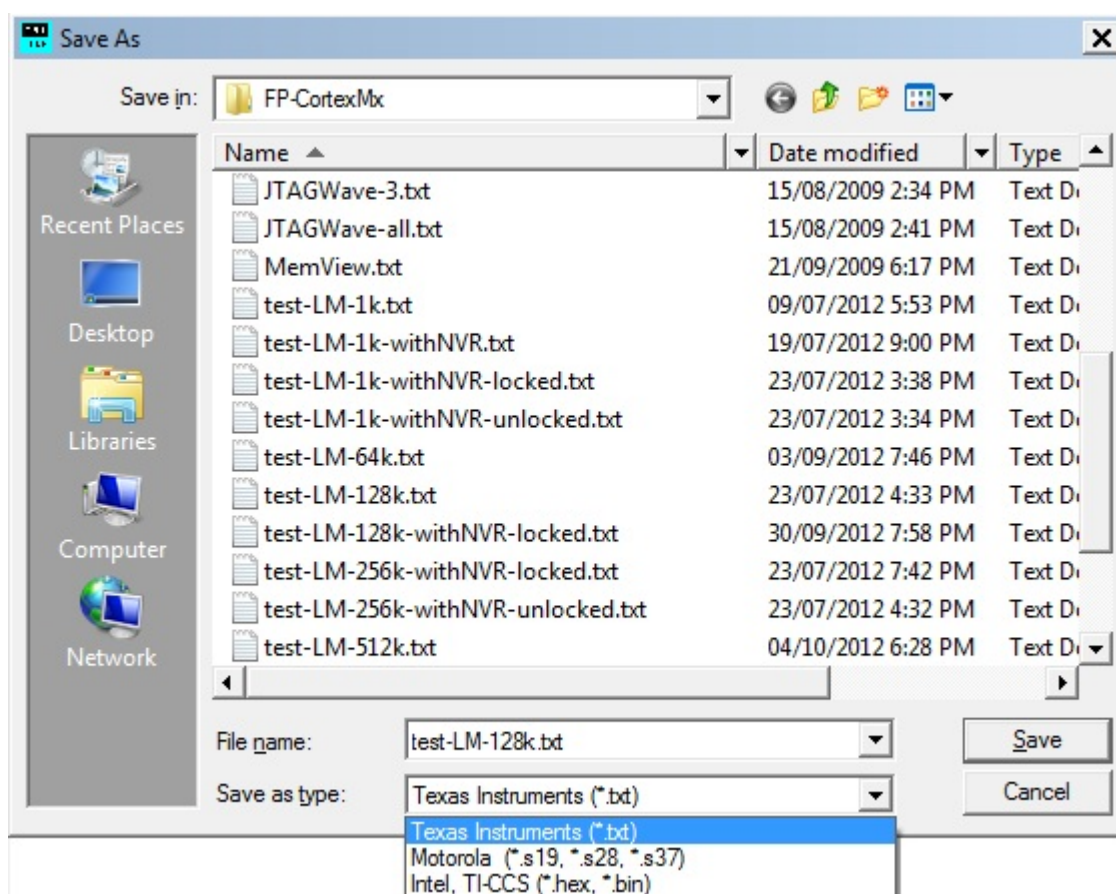
Figure 4.5: Save code file.

4.3   Flash Protection Bits

Flash memory within the LM series devices can be protected against unauthorized program or read access. The FlashPro-LM Programmer software allows the user to program these bits by selecting the "Enable" check-box within the "Memory Protection" area in the top right corner of the main dialogue screen (shown in Figure 4.1). When the "Enable" option is selected, then all protection bits will be programmed during the "Auto Program" procedure after code contents have been programmed and verified correctly. These bits can also be programmed manually by pressing the "Lock Device" button. When programming Flash Protection Bits it is necessary to power-cycle the target device to verify that these bits have been successfully committed (see Section 4.4).

   Although some devices allow the user to reset these protection bits to factory settings using the "Clear Locked Device" procedure, not all devices support this feature.   **On most devices protection bits cannot be re-programmed once committed** . Therefore it is important to verify if the recovery procedure is supported on the target MCU by checking the relevant datasheet from Texas Instruments if you intend to program protection bits multiple times.

   Some processor revisions are known to contain bugs that prevent memory protection bits from being committed (e.g. see LM3S3748 RevA0 errata). In some cases you can still disable debug access because the debug register commits successfully, consult MCU errata on the Texas Instruments' website regarding the revision you are using. When the FlashPro-LM Programmer detects that protection bits have not been programmed successfully, but the user has requested to program debug bits, a popup message will ask if programming should proceed anyway (see Figure 4.6). Normally, the debug register is programmed last only if all other operations are successful.

   The desired configuration of protection bits can be chosen in the **Setup→Memory Protection** dialogue screen (shown in Figure 4.7). The user can select an address range to be protected (for no protection specify both values as 0), or modify individual register bits. Additional protection can be enabled by totally disabling debug access to the MCU. After debug access is disabled then JTAG can no longer be used to communicate with the target device. All these protection bits can either be chosen from the dialogue screen, or taken from a code file.
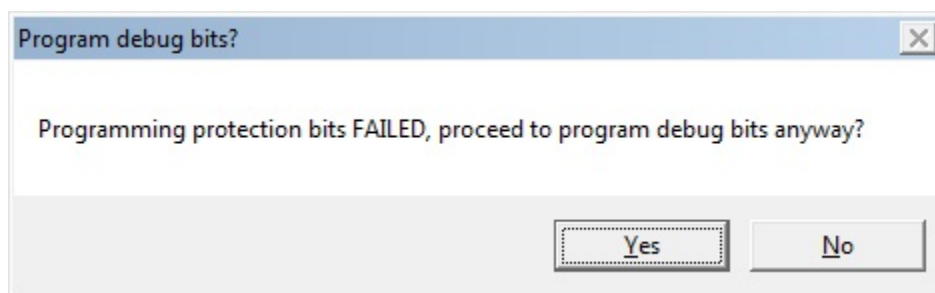
Figure 4.6: The programmer will ask the user if they wish to program the debug register even if protection bits haven't been programmed successfully. This is useful in overcoming some processor bugs that prevent protection bits from being committed.

Figure 4.7: Memory Protection.

4.4 Power Device from Adapter

The programming adapter is powered from the USB Port interface. A target device can be powered from the programming adapter with voltage range from 2.2V to 3.6V in 0.1V steps selected in the voltage selector located in the "MCU Vcc" box (see Figure 4.8).

The target device will be powered from the adapter if the check-box "Enable" is selected. By clicking the POWER ON/OFF button you can also turn the power on or off manually on the target device. Current DC voltage on the target device is continuously monitored and displayed in the "MCU Vcc" box, even if the target device is powered from an external DC source. If DC voltage is higher than 2.0 V, then a yellow box will be displayed, indicating that DC voltage is OK and target device is fully functional under this DC voltage. If DC level is below 2.0V, then an access denied sign box will be displayed (red sign with white line). If DC level is below 1V, then blank sign box will be displayed.

When the target device is powered from an external power supply then the check box "Enable" should not be selected. RESET button located on the right side on the POWER ON/OFF button (Figure 4.1) can generate a reset pulse to the target device. Pressing this button will reset the target device manually at any time.

When programming Flash Protection Bits it is necessary to power-cycle the target device to verify that these bits have been successfully committed. The programmer will do this automatically when power is taken from the adapter. However, if power is taken from an external power source and the **"Enable" check-box is disabled** in the "MCU Vcc" area, a **popup message will wait for the user to power-cycle the device before proceeding with verification (shown in Figure 4.9)** . Some processor revisions are known to contain bugs that prevent memory protection bits from being committed (e.g. see LM3S3748 RevA0 errata). In some cases you can still disable debug access because the debug register commits successfully, consult MCU errata on the Texas Instruments' website regarding the revision you are using. When the FlashPro-LM Programmer detects that protection bits have not been programmed successfully, but the user has requested to program debug bits, a popup message will ask if programming should proceed anyway (see Figure 4.6). Normally, the debug register is programmed last only if all other operations are successful.
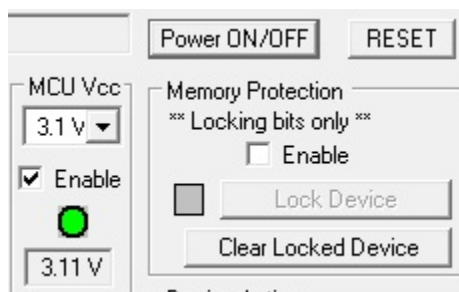
Figure 4.8: Power device.
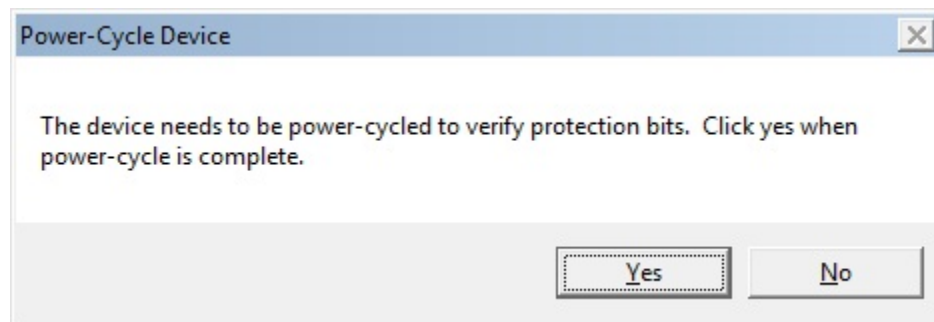
Figure 4.9: Power-cycle required to begin verification.

4.5 Target Device Action Result

After any programming action is performed, a result icon will be displayed next to the button pressed indicating the result status. Some actions, like "Auto Program", perform many tasks and each part of the result is displayed in the test result icons (see Figure 4.10). Each indicator can assume any of the following conditions:

- ☐ - idle status.
- ⬤ - Test in progress. For power on/off - DC voltage is correct.
- ⬤ - access enabled.
- ⛔ - access denied. For power on/off - DC voltage is too low (below 2.0V).
- ☑ - device action has been finished successfully.
- ☒ - device action has been finished, but result failed.
- ☒ - applies to blank check only. Memory is not clean, but the specified memory segment is.

Figure 4.10: Check boxes will indicate the statue of each operation during programming.

4.6   Device Action Box

The Device Action box contains 8 buttons (see Figure 4.11). Each button allows a specific action to be executed. Software procedures related to each action allow you to fully execute the desired task, without the need to follow a specific sequence of actions. Every action starts by powering up the target device, if Power Device from the Adapter is enabled. When the DC voltage level becomes higher then 2.0V, communication with the target device is initiated via the debug interface. If the debug interface is disabled for any reason (i.e. debug register has been programmed to disable debug interface) all these actions will fail. Alternatively, when the debug interface is enabled, and the specified action is completed successfully, a green check marks will appear (see Figure 4.11). The PC application returns to original state after each operation, meaning that as far as the application is concerned the order the buttons are pressed in doesn't matter.

Progress of all actions is displayed in the report window. If the particular action has been finished successfully, then message "done" or "OK" will appear on the right side of processed procedure (Figure 4.12). If not, a message "failed" will be displayed and selected action will be terminated. Final status is also displayed in the Status window (see Figure 4.13) as Active (blue), Pass (green), or Fail (red). On the bottom of the programmer dialogue screen the progress bar is displayed and the total run time is shown in the report window. Run time does not include the time when user interaction is required.
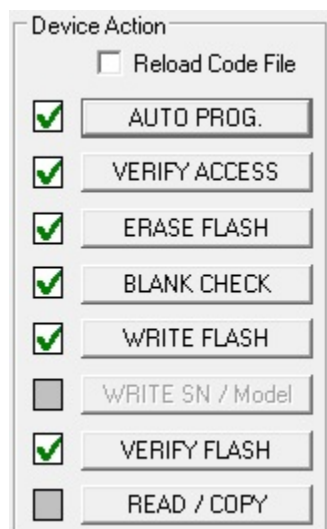
Figure 4.11: Actions that can be taken using the Programmer.
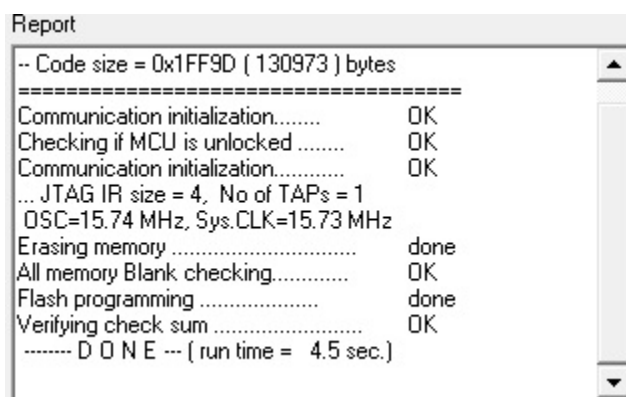
Figure 4.12: This text window shows text messages describing the actions being performed, and how long they take.

Figure 4.13: A summary of the entire programming procedure.

4.6.1   Auto Program

The "Auto Program" button is the most frequently used button when programming a target microcontroller device in a production process. The "Auto Program" button activates all required procedures to fully erase, program and verify flash memory contents. "Auto Program" executes the following procedures:

- reload code file when "Reload Code File" is selected (useful for debugging when the code file is frequently modified),

- initialization,

- read retain data from the flash if specified (optional),

- read label information (Serial Number, Model, Group, Revision)(optional),

- erase flash memory,

- erased memory blank check,

- flash programming and verification,

- assign or retrieve label information (optional),

- restore retain data (optional),

- retain data verification (optional),

- assigned or retrieved label verification (optional),

- flash memory check sum verification (optional),

- set flash protection bits (optional).

In the report window you can see report messages during the "Auto Program" procedure (see Figure 4.12).

Status window (see Figure 4.13) has a counter that is useful in a production process. The total number of programmed devices can be entered in the Total edit line. The Balance line shows the

number of devices that have not been programmed yet. The Balance counter is initialized to the value entered in the Total edit line and is decremented every time "Auto Program" is completed successfully. In the bottom box in the Status group is displayed the number of available serial numbers taken from a user specified serial numbers file.

Note:

Balance counter works only with the "Auto Program" procedure.

4.6.2 Verify Access

This button allows the user to check that the FlashPro-LM Programmer has access to the target device.

4.6.3 Erase Flash

The "Erase Flash" function erases the selected flash memory segments, or mass (all) memory. If any option other then "Erase All Memory" is selected in the Memory Options Setup (see Chapter 6 for configuration details), then a message shown in Figure 4.14 will be displayed. This message clarifies which segment of flash memory is to be erased because typically a mass erase is recommended.
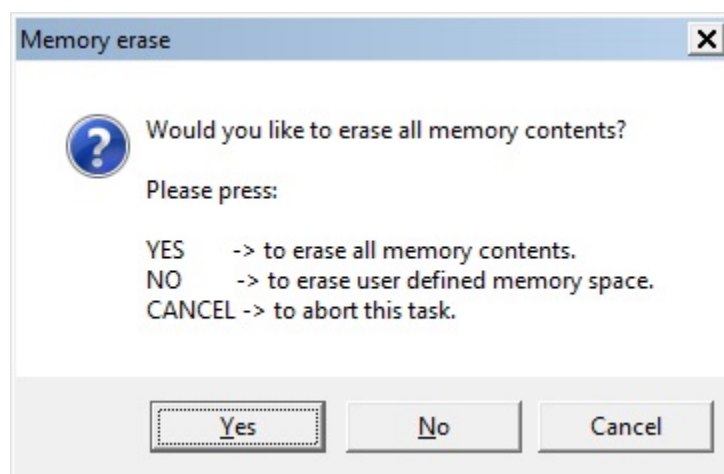
Figure 4.14: This message clarifies which segment of flash memory will be erased.

4.6.4 Blank Check

The "Blank Check" function checks if flash memory of the target microcontroller is blank (all bytes contain the value 0xFF). This test performs two checks: (i) determine if the entire memory contents are clean, (ii) check memory segments specified by the user (see setup in Memory Erase/Write Group). The following conditions can appear at the completion of this operation:

- ☑ - all memory is blank
- ☒ - all memory is not blank, but selected part of it is.
- ☒ - memory is not blank.

4.6.5 Write Flash

The "Write Flash" function writes contents from the code file to flash memory. If this button is pressed for a second time after the target device has already been programmed, then programming will fail because those flash segments are not empty (not 0xFF). Erase and blank check flash segments you wish to program before pressing the "Write Flash" button (see Chapter 6 for configuration details).

4.6.6 Write SN/Model

When the "Write SN/Model" button is clicked, Serialization (see Chapter 8.2) information will be manually programmed to the target device. **It is NOT recommended to use this button outside of the "Auto Program" procedure other than for testing purposes** . Only the "Auto Program" procedure will decrement serial numbers from a serial numbers file (if specified)

and log used serial numbers into the output file. Also, if you chose to select Check Sum Options (see Chapter 9), a check sum that includes the serial number will not be updated if you change the serial number using this button manually.

4.6.7 Verify Flash

The "Verify Flash" function compares the contents of flash memory with data from the code file. This function can use the standard memory verification method (byte by byte) or verify the calculated check sum of the code and check sum of the flash contents (see Chapter 6).

<div align="center">

Note:

</div>

During verification, either all memory or just the selected part of memory is verified, depending on settings specified in the Memory Erase/Write Address Range in the Memory Options setup. See Chapter 6 for details.

4.6.8 Read/Copy

The "Read/Copy" function reads data from the target microcontroller and displays it in the Flash Memory Data window (see Figure 4.15). This window can also be opened by selecting the **View→Flash Memory Data** option. Flash memory data viewer, shown in Figure 4.15, displays the code address on the left side, data in hex format in the central column, the same data in ASCII format in the right column. Contents of the code viewer can be converted to Texas Instruments *.txt file format by clicking on the "Convert to TI format" button. Data will be viewed in the Notepad Editor.

The address range to be displayed in the Flash Memory Data window can be specified in the Memory Options screen. See Chapter 6 Read group for details. When the "Copy" button is clicked, then contents of the read target device memory will be saved in the specified by user file name and opened as a current Code File. As a consequence of this action, programmer setup will be modified for the copy procedure. Serialization will be disabled and the "All Memory" option will be selected in the "Write/Erase/Verify Address Range". The message shown in Figure 4.16 will be displayed. When the button "OK" is pressed then the programmer is ready to program the target device.
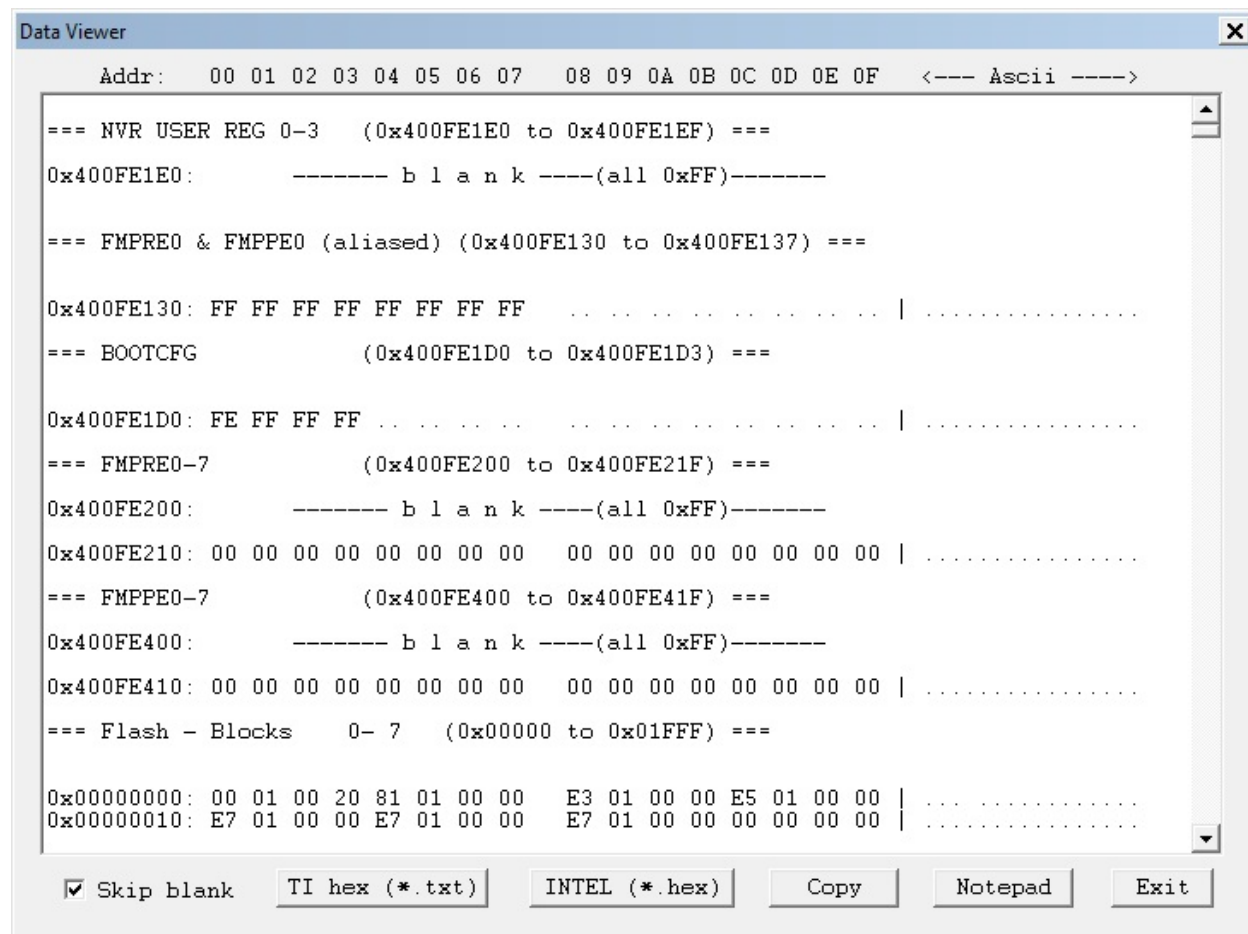
Figure 4.15: Flash memory contents as read from the MCU.
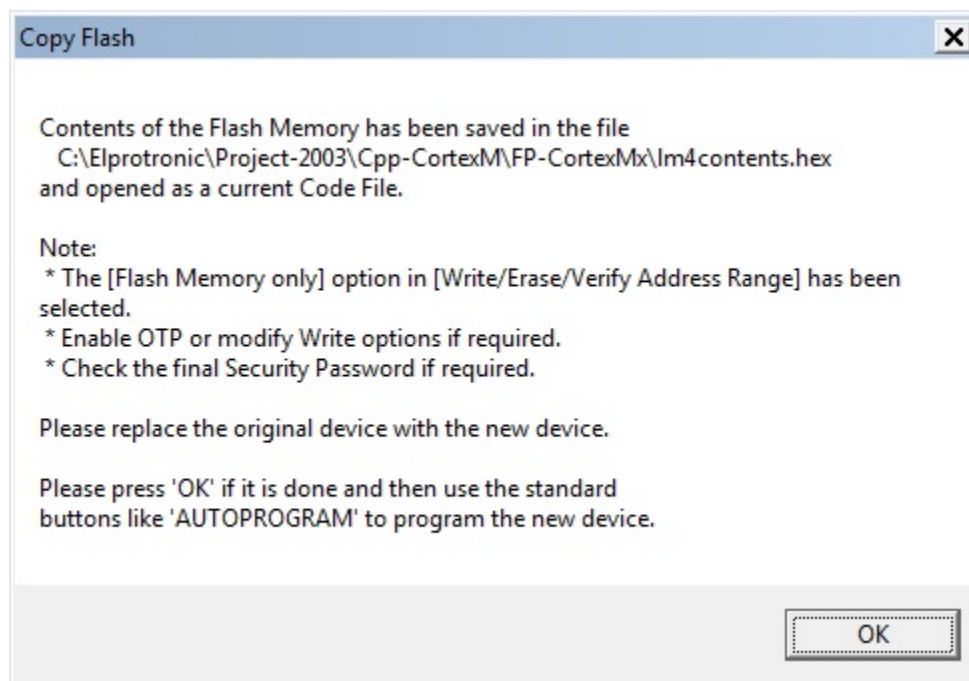
Figure 4.16: Contents of flash have been copied to a file and can be used as a new Code file.

4.7   Next Button

The "Next" button is a dynamically programmable device action button (use shortcut function key F5 to operate). At start-up the "Next" button is disabled (see Figure 4.17, left), but when any button from the Device Action group is pressed, the "Next" button will take on the name and feature of that button. For example, if "Auto Program" has been used then its name will be displayed on the "Next" button (see Figure 4.17, middle) and it will perform the same function as the "Auto Program" button . The "Next" button will retain this functionality until a different Device Action button is pressed. For example, if the button"Read Flash" is clicked, then from this moment on the button "Next" will take the name and function of the "Read Flash" button and so on (see Figure 4.17, right). The purpose of the "Next" button is to perform the same task repeatedly on a series of target MCUs during production.
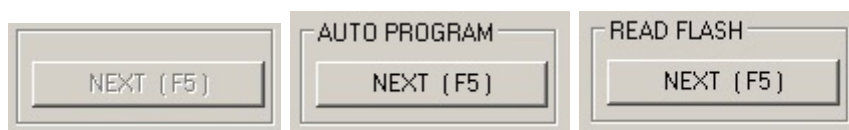
Figure 4.17: The next button can take on multiple functions depending on which Device Action was used previously.

Chapter 5

Data Viewers

Data from code files and flash memory can be viewed and compared in data viewers. Contents of a selected code file can be viewed by selecting the **View→Code File Data** option. The Code data viewer, shown in Figure 5.1, displays the code address on the left side, data in hex format in the central column, and the same data in ASCII format in the right column. Data in hex format is displayed from 0x00 to 0xFF for addresses corresponding to the code file. Data from other addresses is displayed as double dots **..** . If code size exceeds flash memory size in the selected microcontroller, a warning message will be displayed first. The contents of the code viewer can be converted to TI (*.txt) or Intel (*.hex) file format by clicking on the **TI hex** or **INTEL** button.

Contents of **Flash Memory Data** can be viewed by selecting the **View→Flash Memory Data** option. To be able to see flash memory contents, the **Read/Copy** button must be used first (as described in Section 4.6.8). The **Flash Memory Data** viewer displays memory addresses, data in hex and ASCII format in the same way as the code data viewer shown in Figure 5.1.

Contents of the code file and flash memory can be compared and differences can be displayed in a the viewer by selecting the **View→Compare Code & Flash Data** option. Only data that are not the same in the code file and flash memory will be displayed. The first line displays code file data, and the second line displays flash memory data as shown in Figure 5.2. If all the aforementioned data are identical, then a **No difference found** message will be displayed on the screen.
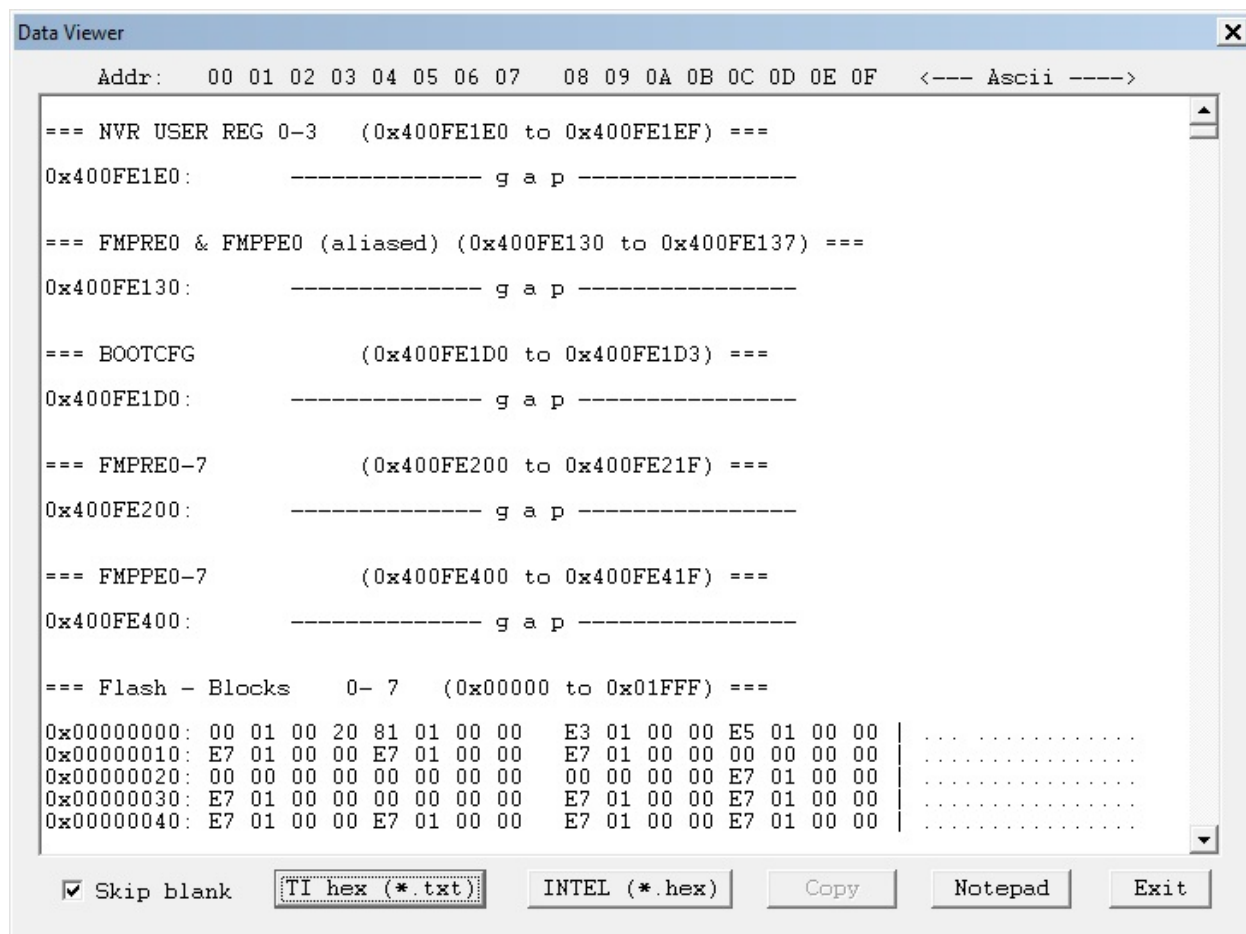
Figure 5.1: Code File Data. The selected option on the bottom ignores all bytes that have the value of **0xFF** , which represents empty bytes.

```
Data Viewer                                                                              ×

   Addr:    00 01 02 03 04 05 06 07   08 09 0A 0B 0C 0D 0E 0F   <--- Ascii ---->

      :    =========== Data Result Comparation.  =============
      :    == Only data from the code file that does not match==
      :    ==   with the Flash Memory data is displayed.     ==
      :    ==   Extra data from the Flash Memory is ignored.  ==

    : First Line  - Data from the Code File.
    : Second Line - Data from the Flash Memory.

0x00000000: 00 01 00 20 81 01 00 00   E3 01 00 00 E5 01 00 00 | ................
0x00000000: FF FF FF FF FF FF FF FF   FF FF FF FF FF FF FF FF | ................

0x00000010: E7 01 00 00 E7 01 00 00   E7 01 00 00 00 00 00 00 | ................
0x00000010: FF FF FF FF FF FF FF FF   FF FF FF FF FF FF FF FF | ................

0x00000020: 00 00 00 00 00 00 00 00   00 00 00 00 E7 01 00 00 | ................
0x00000020: FF FF FF FF FF FF FF FF   FF FF FF FF FF FF FF FF | ................

0x00000030: E7 01 00 00 00 00 00 00   E7 01 00 00 E7 01 00 00 | ................
0x00000030: FF FF FF FF FF FF FF FF   FF FF FF FF FF FF FF FF | ................

0x00000040: E7 01 00 00 E7 01 00 00   E7 01 00 00 E7 01 00 00 | ................
0x00000040: FF FF FF FF FF FF FF FF   FF FF FF FF FF FF FF FF | ................

0x00000050: E7 01 00 00 E7 01 00 00   E7 01 00 00 E7 01 00 00 | ................
0x00000050: FF FF FF FF FF FF FF FF   FF FF FF FF FF FF FF FF | ................

0x00000060: E7 01 00 00 E7 01 00 00   E7 01 00 00 E7 01 00 00 | ................
0x00000060: FF FF FF FF FF FF FF FF   FF FF FF FF FF FF FF FF | ................

0x00000070: E7 01 00 00 E7 01 00 00   E7 01 00 00 E7 01 00 00 | ................
0x00000070: FF FF FF FF FF FF FF FF   FF FF FF FF FF FF FF FF | ................

0x00000080: E7 01 00 00 E7 01 00 00   E7 01 00 00 E7 01 00 00 | ................

 ☑ Skip blank   TI hex (*.txt)    INTEL (*.hex)    Copy     Notepad    Exit
```

Figure 5.2: Comparison of code and flash memory data from the target processor.

Chapter 6

Memory Option Dialogue Screen

The Memory Options dialogue screen selected by choosing the **Setup→Memory Options** option (shown in Figure 6.1) has four settings groups and one information group. These settings groups allow the user to specify memory segments for erase, write, verify and read operations. The user can also choose to retain data in flash memory between successive "Auto Program" and "Erase" operations and choose the type of write verification. The information group contains specifications of the selected MCU, such as the address range for non-volatile registers, flash size, and RAM size.

6.1   Memory Erase, Write, and Verify

The "Auto Program", "Erase", "Write", and "Verify" operations used in the main dialogue screen (shown in Figure 4.1) use addresses specified in this dialogue screen. This settings group has seven options:

1. **Unlock [Auto Program only]** - When this option is selected the "Auto Program" operation will attempt to unlock the target device using the "Clear Locked Device" recovery procedure IF the FlashPro-LM Programmer fails in gaining access. The recovery procedure is only used in this context if the debug interface is disabled, not if selected memory pages are protected.

2. **Update only** - When this option is selected the "Auto Program" operation will not erase memory contents. Instead, contents of code data taken from the code file will be downloaded to flash memory. This option is useful when a relatively small amount of data, such as calibration data, needs to be added to flash memory. Other address ranges should not be included in the code file, meaning that the code file should contain ONLY the data which is to be programmed to flash memory. For example, if the code file contains data as shown in TI format:

```
@1008
25 CA 80 40 39 E3 F8 02
@2200
48 35 59 72 AC B8
q
```

Figure 6.1: Control how memory will be accessed using this dialogue window.

Then 8 bytes of data will be written starting at location 0x1008 and 6 bytes of data starting at location 0x2200. The specified addresses should be blank before writing (contain a value of 0xFF). Before the writing operation is actually performed, the FlashPro-LM Programmer will verify automatically if this part of memory is blank and will only proceed to program the device if verification is successful.

Even Number of Bytes.

The number of bytes in all data blocks must be even. Words (two bytes) are used for writing and reading data. In case that the code file contains an odd number of bytes, the data segment will be appended by a single byte containing a blank value of 0xFF. This value will not overwrite the current memory contents (because **Update only** is selected), but verification will fail if the target device does not contain a blank value of 0xFF at that location.

3. **NVR Memory only** - If an MCU contains a non-volatile register (NVR) segment then it will be programmed. NVR segments include user registers (USER_REGS) and protection registers (FMPPE and FMPRE). Values for user registers can only be specified in the code file; however, values for protection registers can be specified in the code file, or manually by using the **Setup→Memory Protection** option (see Section 4.3). NVR segments cannot be erased normally, only the "Clear Locked Device" procedure can reset these registers to factory settings (not all processors support this feature).

4. **NVR and Flash Memory** - This is the most frequently used option during programming. All memory is erased before programming, and all contents from the code file are downloaded to the target microcontroller's flash memory. If the MCU contains an NVR segment then it will be programmed.

5. **Flash Memory only** - All flash memory will be erased and programmed. Contents of NVR memory will be ignored.

6. **Used by Code File** - This option allows main memory segments and NVR segments to be modified when specified by the code file. Other flash memory segments are not touched. This option is useful if only some data, like calibration data, needs to be replaced.

7. **User defined** - This option is functionally similar to options described before, but memory segments are explicitly chosen by the user. When this option is selected, then on the right side of the memory group check boxes and address edit lines will be enabled. The check boxes allow the user to select memory segments to be enabled (erased, programmed, verified). Edit lines in the Main Memory group allow the user to specify the main memory address range (start and stop addresses). The start address should specify the first byte in the segment, and the stop address should specify the last byte in the segment (last byte is programmed). Since the main memory segment size is 0x400, then the start address should be a multiple of 0x400. The stop address should specify the last byte of the segment to be written.

6.2   Read

The Read Address Range group (see Figure 6.1) specifies the address range used when the "Read/Copy" operation is used. Memory read setup has four options available:

1. NVR Memory only,

2. NVR and Flash Memory,

3. Flash Memory only,

4. User defined.

The meaning of each option is the same as for the Memory Erase, Write, and Verify settings.

6.3   Retain Data in Flash

This feature allows the programmer to preserve flash contents between successive programming and erase operations. By specifying the start and end addresses (inclusively), the selected bytes will be first read and then copied to an internal buffer before the selected flash memory is erased. After the memory segment coinciding to this region has been erased, the internal buffer will be used to program this segment of flash memory to its previous values. NVR addresses cannot be entered in this section.

6.4   Write Verification

Verification setup allows the user to select one of three write verification methods:

1. **Fast Verification** - Each byte is verified after being written, and at the end of the process a check sum is calculated based on current flash memory and compared to the calculated check sum taken from the code file.

2. **Standard Verification** - Includes Fast Verification as well as reading the contents of the entire flash after programming to compare them against the code file. If they are the same, then verification is successful. Typically, the standard verification procedure requires the same amount of time as the read/write procedure.

3. **None** - Not recommended. Some MCUs have flash with an extremely limited number of program/erase cycles (e.g. 100 instead of the usual 1 million), meaning that after that number is exceeded programming is no longer reliable (check MCU errata from TI website). Verification will check that the code file specified has actually been copied to the target device.

Chapter 7

Adapter Options

7.1 Communication Speed

The default communication speed between the programming adapter and target device is 1 Mb/s. Under some conditions, for example when the cable between FPA and target device is long or some protection components are installed in the debug interface, fast communication cannot be used. In this case the lower speed of 0.2 Mb/s can be used to establish communication between FPA and target device (see Figure 7.1 communication speed selector).
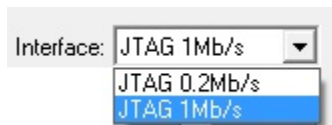
Figure 7.1: Use this drop-box to select the desired communication speed.

7.2   Reset Dialogue Screen

The Target's Reset Options dialogue screen enables the user to select the reset pulse duration and reset line state at the end of the programming process (see Figure 7.2).

7.2.1   Reset pulse duration

The reset pulse allows the adapter to initiate communication with a microcontroller using the debug interface. In most cases a pulse width of 10ms is sufficient to initiate communication; however, if there is additional load on the reset line a longer pulse might be required. Therefore, four additional settings, 100, 200, 500 ms and custom, are available. When a RESET IC circuit is used then the custom defined reset pulse duration should be used. Two parameters of the custom reset pulse are defined - initialization reset pulse time (typically very short - 1 ms) and an idle reset time. Idle reset time must be set at least to the duration of the reset time generated by the RESET circuit.
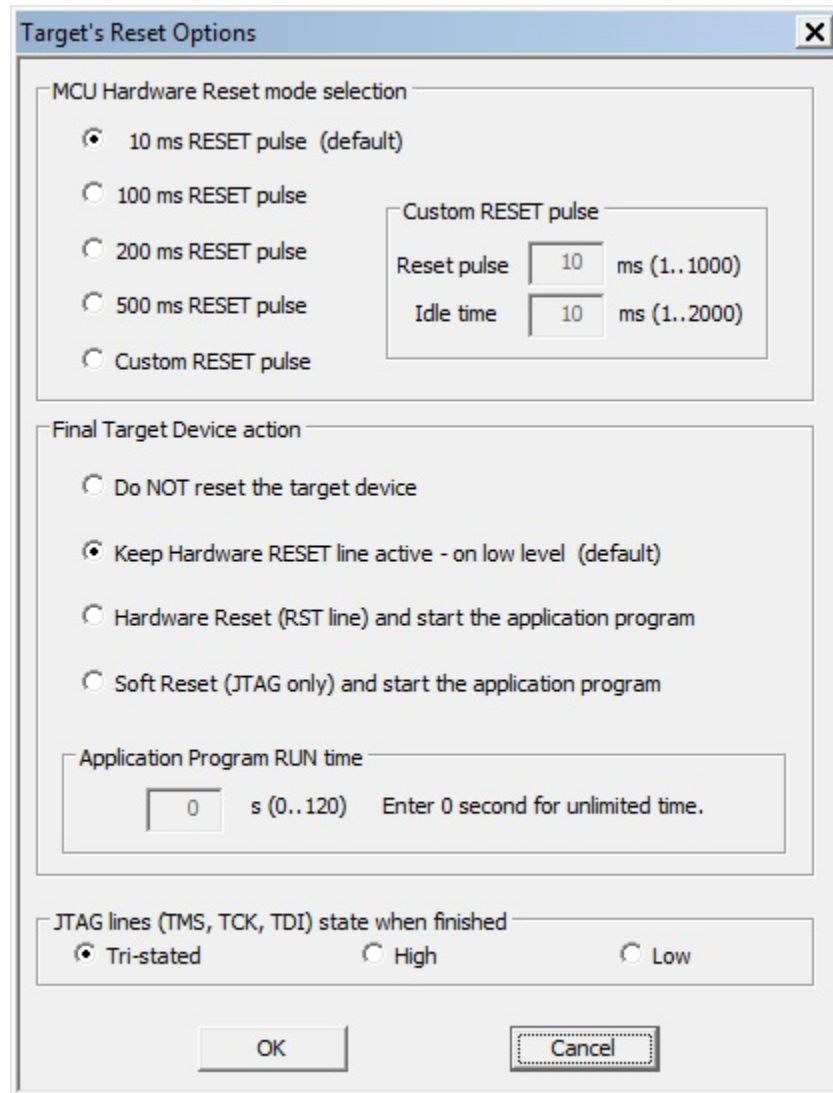
Figure 7.2: Use this dialog window to configure reset options.

7.2.2 Final Target Device action

Every device action, like "Auto Program" or "Read/Copy" starts with the activation of the RESET line (active low). When the device programming action begins the RESET line is raised high. When device action is finished, the RESET line is again asserted, protecting the target device from running the application program. This method is commonly used to protect the programming adapter from DC overload. However, when target device is supplied from its own power supply, or a battery, then overload protection of the programming adapter is no longer necessary.

The target device can be set to run an application immediately after the target device is programmed. This permits verification of the programmed device if required. To do this check the **Hardware Reset (RST line) and start the application program** option. Application run time can be unlimited (enter 0) or limited up to 120 seconds. Limited time is specified in the "Application Program RUN time" box.

7.3   Preferences Dialogue Screen

The Preferences Dialogue screen allows the user to select multiple features, including the option to view history in the report window and to log messages to a file (see Figure 7.3). The **Report History Display Enable** check-box when clicked will allow report history to be displayed for up to 8 kB worth of characters (approximately 20 last communication messages). When disabled, then only last programming report is displayed. Enabling the log file will save history messages to a log file, preserving it for later analysis if necessary. To control the size of each log file, the user can choose to create a new log file every day, or every month.

All programming actions at the end can generate the Beep OK tone. When a lot of units are programmed then the beeping can become a nuisance and therefore it can be disabled. However, the error programming tone is enabled permanently and cannot be disabled because it should not be ignored.

It is also possible to disable multiple warning messages for a single programmer operation, which is useful when the user is aware of the problem and simply wishes to proceed. This can normally happen when attempting to read protected memory pages, at which point the programmer will give a warning message indicating a read error per page. Enabling this option will configure the programmer to only display the first warning message.
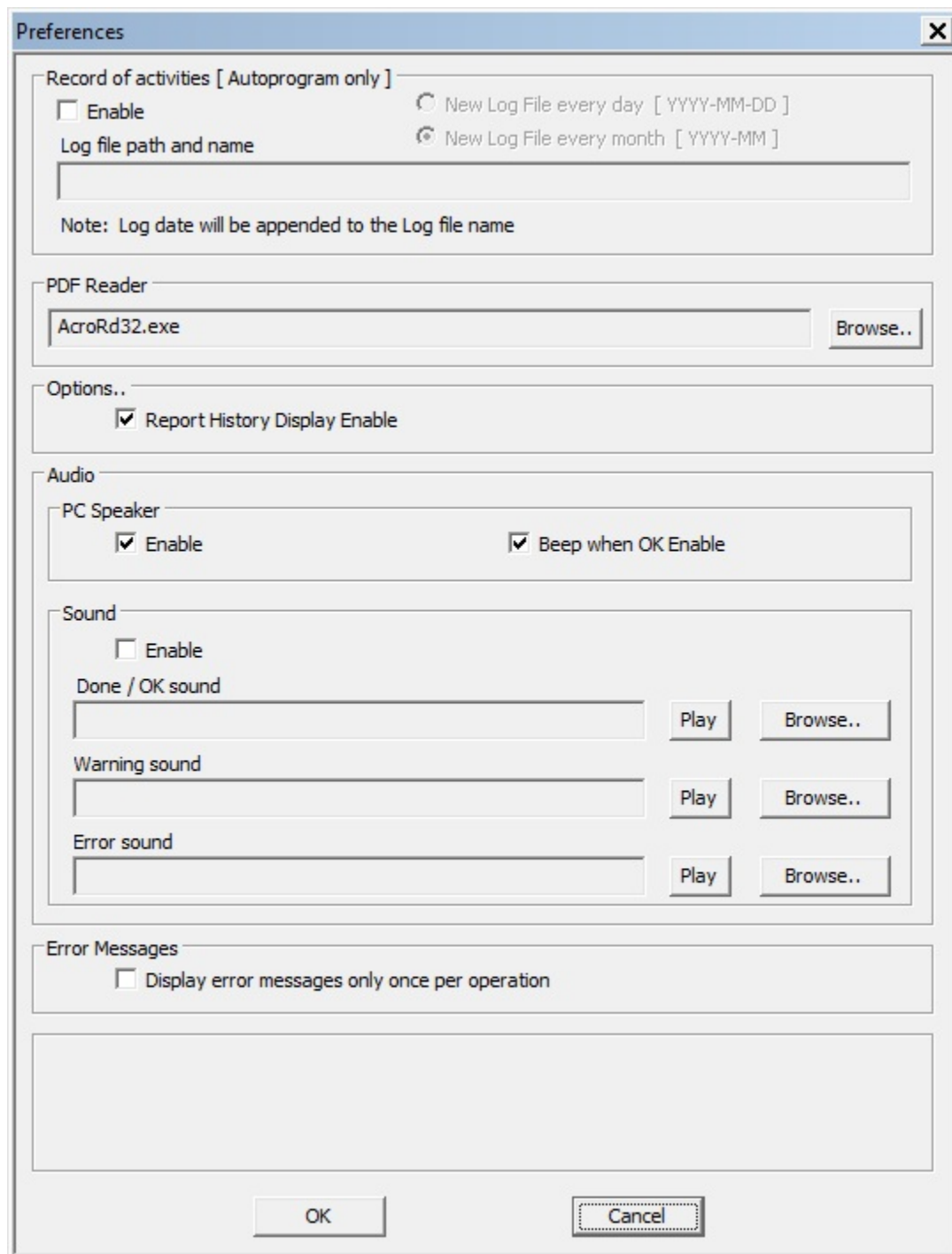
Figure 7.3: Use this dialog window to configure your preferences.

Chapter 8
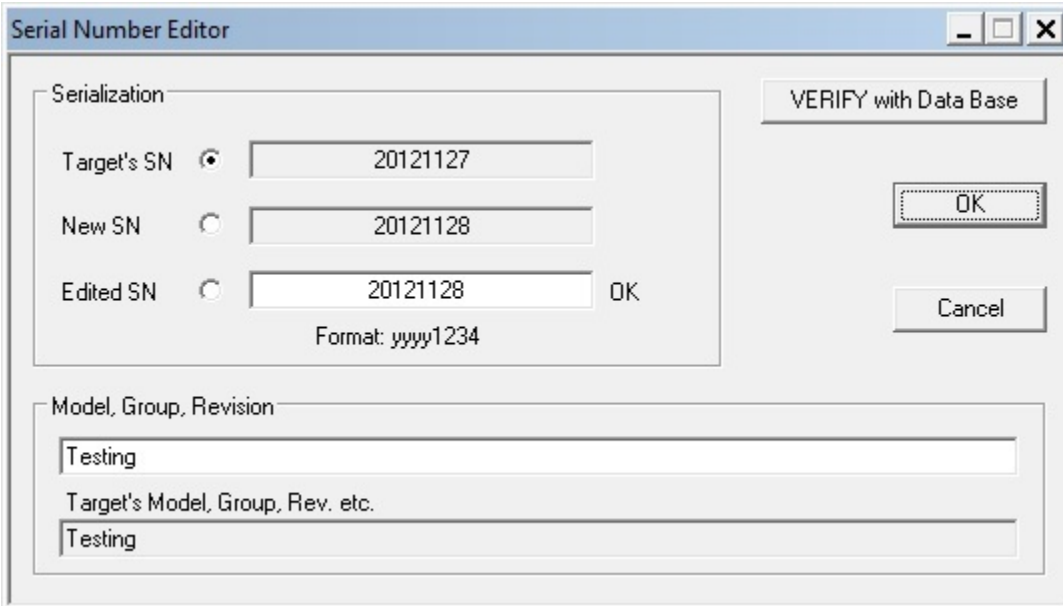
Serialization

The FlashPro-LM Programmer has the ability to automatically program the target device's serial number (SN) and save it in flash memory. New SNs can be created automatically by incrementing a starting base SN, or can be taken from a file created by the user. SNs that have been used are stored in separate data files to account for devices that have already been programmed. In addition to this SN, the model name, group, and revision number can also be programmed to the target device. The user can specify the display SN format and location in flash memory where this information will be stored.

The SN is programmed when "Auto Program" or "Write SN/Model" buttons are pressed and the SN feature is enabled. When the "Auto Program" function is activated the SN is programmed to the target's device memory at the same time as code data. If "Auto Program" fails for any reason then the new SN is neither created nor programmed to the device.

The software also allows the microcontroller to retain its SN if one has already been assigned to it. Every time a device is programmed and serialization is enabled the contents of the target's memory are scanned for an existing SN. If the SN is found at the specified location the message in Figure 8.1 will appear and allow the user to decide to keep the old SN, program a new SN from the list, or program a SN typed manually.

Figure 8.1: If an MCU already contains serialization information at a specified location then a conflict window will appear giving the user the option to retain the old serial number, or program in a new one.

8.1   Serialization Dialogue Screen

The Serialization Dialogue screen, shown in Figure 8.2, allows the user to configure how serialization is done. Serialization can be enabled, or disabled, by selecting the check mark in the **ENABLE Serialization** check-box. When serialization is disabled all edit lines and check boxes are disabled. When serialization is enabled, most fields have to be set to ensure that the result will be as the user expects. The necessary fields include:

- **Display Format** - The display format describes the meaning of the characters that comprise the SN.

- **Memory Format** - The memory format describes how the SN will be stored in the device's flash memory.

- **Memory Location** - This address is the memory location reserved for the SN. It can only overlap code regions if the **Remove code contents in the location where the serialization and model.. are defined** check-box is also selected.

- **Starting Number and Increment** - When using all display formats except "Custom" and "from File", these parameters are necessary. The starting number will indicate the value of the first SN created. Subsequent SNs will be determined by adding the increment to the current SN number. Both of these parameters can only be positive integers and create SNs that remain within the maximum range described later in this chapter.

- **Output File** - The specified file will store all successfully programmed SNs.

The serialization dialog screen also contains several useful optional features:

- **Model/Group/Revision** - In addition to the aforementioned SN formats, the software also allows the user to program each device with separate HEX or ASCII data. This information is saved at a different memory location from the SN, and does not change after each programming cycle. This is used to program static information that should be the same for each target device, like the software revision number, model, etc.

- **BarCode Scanner** - The BarCode Scanner can be used as an input mechanism for new SNs. When enabled, scanning a barcode will set the "Next SN" shown in Figure 4.1 to the value read by the BarCode Scanner. For convenience, the software can be configured to start the "Auto Program" procedure immediately after the BarCode Scanner reads a new number, by selecting the **Start AUTOPROGRAM following BarCode scan** check-box.
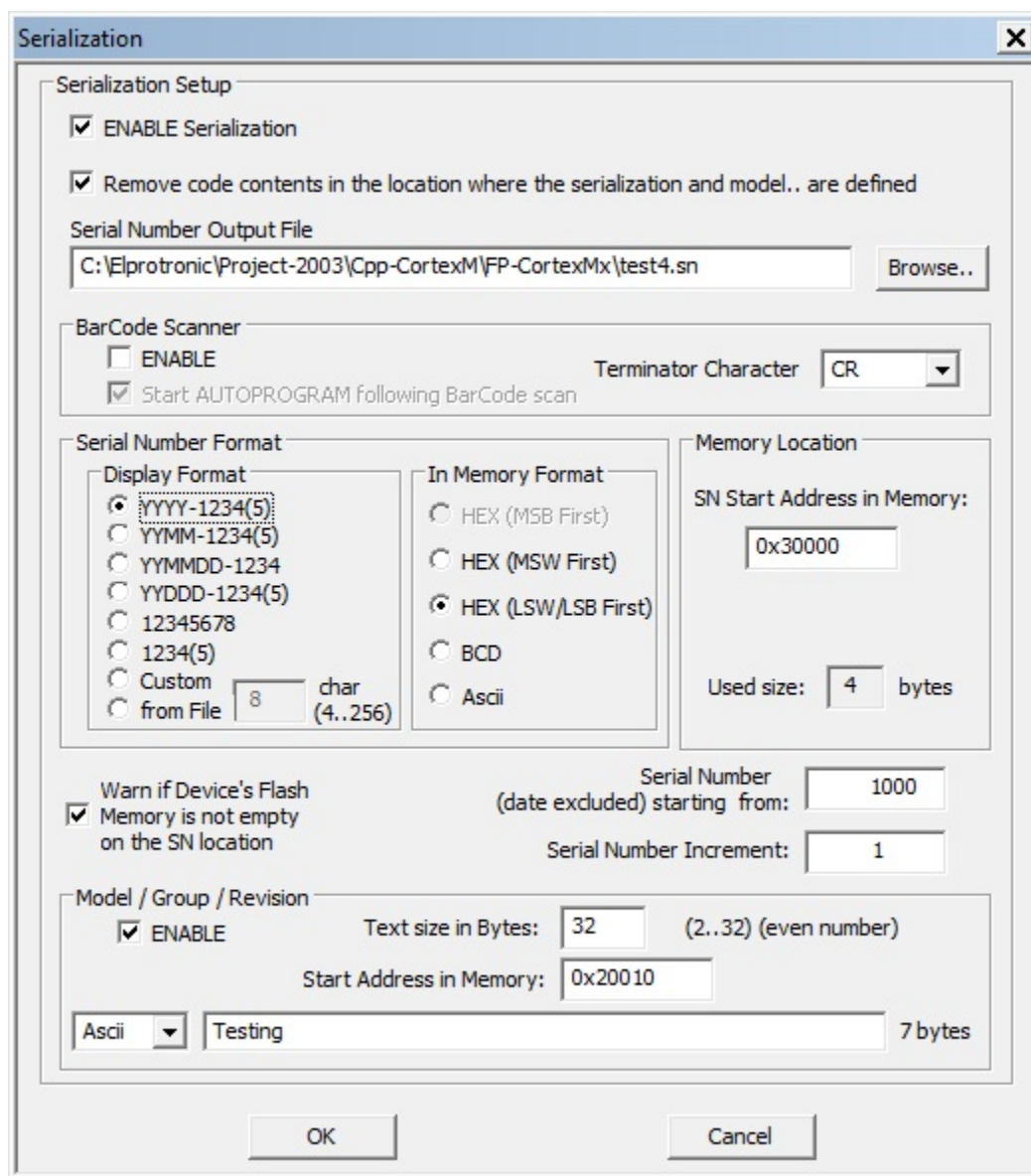
Figure 8.2: Serialization options can be selected here.

8.2   Serial Number Display Format

Programming software has eight formats for writing a serial number, referred to as **Display Format** , and three methods of storing the SN in memory, referred to as **In Memory Format** in the serialization dialogue screen. When a serial number is created, the current date (if required) is taken from the PC timer, therefore ensure that your computer has the correct date and time.

  Display Formats:

1. **YYYY-1234(5)** - (SN Format F0) Serial number has 8 (BCD) or 9 (HEX) characters. First four characters specify the year, and the remaining 4 or 5 characters specify the serial number up to a maximum of 9999 or 65535 respectively, e.g. SN 20039999 or 200365535.

2. **YYMM-1234(5)** - (SN Format F1) Serial number has 8 (BCD) or 9 (HEX) characters. First two characters specify the last two digits of the year, next two characters specify the month, and the remaining 4 or 5 characters specify the serial number up to a maximum of 9999 or 65535 respectively, e.g. 03059999 or 030565535.

3. **YYMMDD-1234** - (SN Format F5) Serial number has 10 characters. First six characters contain date (year, month, day of month) and remaining 4 characters specify the serial number, e.g. 1205309999.

4. **YYDDD-1234(5)** - (SN Format F4) Serial number has 9 (BCD) or 10 (HEX) characters. First five characters contain date (year, day of year from 1 to 366) and remaining 4 or 5 characters specify the serial number up to a maximum of 9999 or 65535 respectively, e.g. 123559999 or 1235565535.

5. **12345678** - (SN Format F2) 8 digit serial number without date stamp. The maximum value is 99999999.

6. **1234(5)** - (SN Format F3) 4 (BCD) or 5 (HEX) digits serial number without date stamp. The 4 or 5 characters specify the serial number up to a maximum of 9999 or 65535.

7. **Custom** - (SN Format F6) 4 to 256 ASCII characters or hexadecimal numbers entered manually or from the Bar-Code Reader.

8. **from File** - (SN Format F7) 4 to 256 ASCII characters or hexadecimal numbers taken from a user created file.

  For SN display formats F0 through F5, the non-date components can be entered using either decimal or hexadecimal notation. Normally only characters from 0 to 9 will be accepted unless a "0x" prefix is added informing the software that a hexadecimal number is begin entered, allowing the user to use A to F as well. All characters are subsequently converted to a chosen memory format, which can be HEX, BCD or ASCII, before they are saved to flash memory.

  When a "Custom" (F6) or "from File" (F7) SN display format is selected then all characters can be accepted depending on the memory format chosen. Specifically, when the ASCII format is selected the SN can be comprised of any characters and is saved using 8-bits per character into

flash memory. Conversely, when the HEX format is selected the SN can only be comprised of HEX characters (0...9,A...F) and is saved using 4-bits per character.

8.3   Serial Number Memory Format

This section describes the different formats in which serial numbers can be stored in flash memory. These numbers can be stored using various types of byte-ordering schemes including:

1. **Most-Significant-Byte (MSB) first** - The most significant byte, in a number composed of multiple bytes, is the byte that represents the greatest value 8-bit sub-component of that number. For example, the number 0x12345678 is composed of 4 bytes: 0x12, 0x34, 0x56, and 0x78, where 0x12 is the most significant byte. When this 4 byte number is stored in memory using MSB first ordering, then the lowest byte address (the first address) will store the most significant byte (0x12).
    Addr: 0x0, Content: 0x12
    Addr: 0x1, Content: 0x34
    Addr: 0x2, Content: 0x56
    Addr: 0x3, Content: 0x78

2. **Most-Significant-Word (MSW) first** - The most significant word, in a number composed of multiple words, is the word that represents the greatest value 16-bit sub-component of that number. For example, the number 0x12345678 is composed of 2 words: 0x1234, and x5678, where 0x1234 is the most significant word. When this 2 word number is stored in memory using MSW first ordering, then the lowest word address (the first address) will store the most significant word (0x1234).
    Addr: 0x0, Content: 0x1234
    Addr: 0x2, Content: 0x5678

3. **Least-Significant-Byte (LSB) first** - The least significant byte, in a number composed of multiple bytes, is the byte that represents the smallest value 8-bit sub-component of that number. For example, the number 0x12345678 is composed of 4 bytes: 0x12, 0x34, 0x56, and 0x78, where 0x78 is the least significant byte. When this 4 byte number is stored in memory using LSB first ordering, then the lowest byte address (the first address) will store the least significant byte (0x78).
    Addr: 0x0, Content: 0x78
    Addr: 0x1, Content: 0x56
    Addr: 0x2, Content: 0x34
    Addr: 0x3, Content: 0x12

4. **Least-Significant-Word (LSW) first** - The least significant word, in a number composed of multiple words, is the word that represents the smallest value 16-bit sub-component of that number. For example, the number 0x12345678 is composed of 2 words: 0x1234, and x5678, where 0x5678 is the least significant word. When this 2 word number is stored in memory
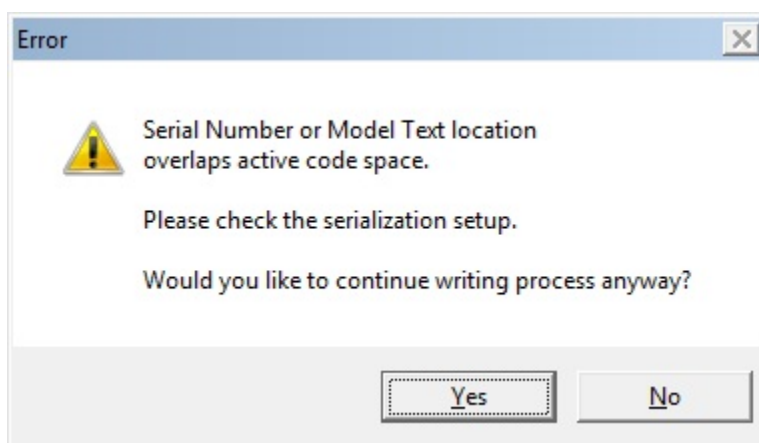
Figure 8.3: This window will appear when there is a conflict between memory allocated to code and the SN. To avoid this pop-up, select the option to "Remove code contents in the location where the serialization and model.. are defined" in Figure 8.2.

using LSW first ordering, then the lowest word address (the first address) will store the least significant word (0x5678).
Addr: 0x0, Content: 0x5678
Addr: 0x2, Content: 0x1234

The location in the target device's flash memory where the SN will be stored is specified in the "Memory Location - SN Start Address in Memory" field of the serialization dialogue screen (see Figure 8.3). Specified address must be even and should be specified in empty memory space, not used by program code or data block. However, if you'd like to integrate the SN with existing code, by programming the SN to a code location, then check the box **Remove code contents in the location where the serialization and model.. are defined** .

The SN memory formats will be described in the following subsections.

8.3.1   HEX Memory Format

When this format is selected the serial number is encoded using hexadecimal (HEX) notation. In general, each sub-component of the SN written in base 10 is converted to an equivalent base 16 number. The number of bytes used varies depending on the display format (F0, F1, F2, F4, and F5 - 4 bytes; F3 - 2 bytes; F6, and F7 - between 2 to 128 bytes).

Display Format: YYYY-12345 - size in FLASH - 4 bytes

Can be stored using MSW, and LSW byte-ordering.

```
SN 200300123 will be encoded as:
YYYY - 2003   ->   0x07D3  (HEX)
```

```
12345 - 00123   ->   0x007B  (HEX)
and combined into one 32-bit HEX number: 0x07D3007B

<-- Addr-->
00-01-02-03

D3 07 7B 00     -> HEX(MSW first)
7B 00 D3 07     -> HEX(LSW first)
```

Display Format: YYMM-12345 - size in FLASH - 4 bytes

Can be stored using MSW, and LSW byte-ordering.

```
SN 030300123 will be encoded as:
YYMM  - 0303   ->   0x012F  (HEX)
12345 - 00123  ->   0x007B  (HEX)
and combined into one 32-bit HEX number: 0x012F007B

<-- Addr-->
00-01-02-03

2F 01 7B 00     -> HEX(MSW first)
7B 00 2F 01     -> HEX(LSW first)
```

Display Format: YYMMDD-1234 - size in FLASH - 4 bytes

Can be stored using MSW, and LSW byte-ordering.

```
The date component is compressed to fit into only two bytes as
follows:

Bit   15-14-13-12-11-10--9  8--7--6--5  4--3--2--1--0
      <---(year-2000)---->  <--month->  <----day---->

SN 0405110123 will be encoded as:
YYMMDD - 040511 ->   0x08AB (HEX)
1234   - 0123   ->   0x007B (HEX)
and combined into one 32-bit HEX number: 0x08AB007B

<-- Addr-->
00-01-02-03

AB 08 7B 00     -> HEX(MSW first)
7B 00 AB 08     -> HEX(LSW first)
```

Display Format: YYDDD-1234 - size in FLASH - 4 bytes

Can be stored using MSW, and LSW byte-ordering.

```
The date component is compressed to fit into only two bytes as
follows:

Bit   15-14-13-12-11-10--9  8--7--6--5--4--3--2--1--0
      <---(year-2000)---->  <----- day of year ----->

SN 041110123 will be encoded as:
```

```
YYDDD - 04111   ->   0x086F (HEX)
1234  - 0123    ->   0x007B (HEX)
and combined into one 32-bit HEX number: 0x086F007B

<-- Addr-->
00-01-02-03

6F 08 7B 00     ->  HEX(MSW first)
7B 00 6F 08     ->  HEX(LSW first)
```

Display Format: 12345678 - size in FLASH - 4 bytes

Can be stored using MSB, MSW, LSB, and LSW byte-ordering.

```
SN 12345678 will be encoded as:
12345678 - 12345678   ->   0x00BC614E  (HEX)

<-- Addr-->
00-01-02-03

00 BC 61 4E     ->  HEX(MSB first)
BC 00 4E 61     ->  HEX(MSW first)
4E 61 BC 00     ->  HEX(LSB first)
4E 61 BC 00     ->  HEX(LSW first)
```

Display Format: 12345 - size in FLASH - 2 bytes

Can be stored using MSB, MSW, LSB, and LSW byte-ordering.

```
SN 12345 will be encoded as:
12345 - 12345   ->   0x3039  (HEX)

<Addr>
00-01

30 39   ->  HEX(MSB first)
39 30   ->  HEX(MSW first)
39 30   ->  HEX(LSB first)
39 30   ->  HEX(LSW first)
```

Display Format: Custom - size in FLASH - between 2 to 128 bytes

Can be stored using MSB, LSB, and LSW byte-ordering. A SN entered using the "Custom" format is entered as a series of either an even number of HEX digits or any number of ASCII characters. This subsection deals only with storing the SN entered as a series of an even number of HEX digits, where two adjacent HEX digits comprise one byte (two digits are adjacent if they have the index j and k, where j=2*i and k=2*i+1, for all i where $0 \leq i < N$, where $N \epsilon$ {2...128}). The SN can be composed of anywhere between 4 and 256 characters.

```
SN sequence entered as ''0x12345678AABBCCDD'' will be stored as:

<--------Addr-------->
00-01-02-03-04-05-06-07
```

```
12 34 56 78 AA BB CC DD -> HEX(MSB first)
DD CC BB AA 78 56 34 12 -> HEX(LSB first)
DD CC BB AA 78 56 34 12 -> HEX(LSW first)
```

Display Format: from File - size in FLASH - between 2 to 128 bytes

Can be stored using MSB, LSB, and LSW byte-ordering. A SN entered using the "from File" format is read from the SN input file and can be a series of either an even number of HEX digits or any number of ASCII characters. Section 8.9 describes how to write a SN input file. This subsection deals only with storing the SN entered as a series of an even number of HEX digits, where two adjacent HEX digits comprise one byte (two digits are adjacent if they have the index j and k, where j=2*i and k=2*i+1, for all i where $0 \leq i < N$, where $N\epsilon$ {2...128}). The SN can be composed of anywhere between 4 and 256 characters.

```
SN sequence entered as ''0x12345678AABBCCDD'' will be stored as:

<---------Addr------->
00-01-02-03-04-05-06-07

12 34 56 78 AA BB CC DD -> HEX(MSB first)
DD CC BB AA 78 56 34 12 -> HEX(LSB first)
DD CC BB AA 78 56 34 12 -> HEX(LSW first)
```

8.3.2 BCD Memory Format

When this format is selected the serial number is encoded using binary-coded decimal (BCD) notation. In general, each digit of the SN written in base 10 is separately expressed using 4 bits such that the resulting HEX number reads like a decimal. This is in contrast to HEX notation where the value of the entire number is converted to base 16. The number of bytes used varies depending on the display format (F0, F1, F2, and F4 - 4 bytes; F5 - 5 bytes; F3 - 2; F6, and F7 - N/A). All BCD serial numbers are stored using MSB first byte-ordering scheme.

Display Format: YYYY-1234 - size in FLASH - 4 bytes

```
SN 20030123 will be encoded as:
YYYY - 2003      ->  0x20 0x03  (BCD)
1234 - 0123      ->  0x01 0x23  (BCD)
and combined into one 32-bit BCD number: 0x20030123

<--Addr--->
00-01-02-03

20 03 01 23      ->  BCD(MSB first)
```

Display Format: YYMM-1234 - size in FLASH - 4 bytes

```
SN 03030123 will be encoded as:
YYMM - 0303      ->   0x03 0x03  (BCD)
1234 - 0123      ->   0x01 0x23  (BCD)
```

and combined into one 32-bit BCD number: 0x03030123

```
<--Addr--->
00-01-02-03

03 03 01 23    ->  BCD(MSB first)
```

Display Format: YYMMDD-1234 - size in FLASH - 5 bytes

```
SN 0405110123 will be encoded as:
YYMMDD - 040511 ->   0x04 0x05 0x11 (BCD)
1234   - 0123   ->   0x01 0x23      (BCD)
and combined into one 32-bit BCD number: 0x0405110123

<----Addr---->
00-01-02-03-04

04 05 11 01 23 ->  BCD(MSB first)
```

Display Format: YYDDD-1234 - size in FLASH - 4 bytes

```
The date component is compressed to fit into only two bytes as
follows:
Bit   15...12   - Year number - multiple of ones     (9,8,...1,0)
      11,10     - Year number - multiple of tens     (3,2,1,0)
      9, 8      - Day number  - multiple of hundreds (3,2,1,0)
      7...4     - Day number  - multiple of tens     (9,8,...1,0)
      3...0     - Day number  - multiple of ones     (9,8,...1,0)

SN 041110123 will be encoded as:
YYDDD - 04111   ->   0x41 0x11  (BCD)
1234  - 0123    ->   0x01 0x23  (BCD)
and combined into one 32-bit BCD number: 0x41110123

<--Addr--->
00-01-02-03

41 11 01 23    ->  BCD(MSB first)
```

Display Format: 12345678 - size in FLASH - 4 bytes

```
SN 12345678 will be encoded as:
12345678 - 12345678  ->  0x12 0x34 0x56 0x78 (BCD)
and combined into one 32-bit BCD number: 0x12345678

<--Addr--->
00-01-02-03

12 34 56 78    ->  BCD(MSB first)
```

Display Format: 1234 - size in FLASH - 2 bytes

```
SN 1234 will be encoded as:
1234 - 1234      ->   0x12 0x34  (BCD)
and combined into one 16-bit BCD number: 0x1234
```

```
<Addr>
00-01

12 34          ->  BCD(MSB first)
```

Display Format: Custom - N/A

Display Format: from File - N/A

8.3.3 ASCII Memory Format

When this format is selected the serial number is encoded using American Standard Code for Information Interchange (ASCII) notation. In general, each character of the SN is assumed to be an ASCII character and converted to its byte-code representative using the standard ASCII table (each character of the SN will occupy 1 byte). The number of bytes used is directly proportional to the number of characters used in the display format (F0, F1, and F2 - 8 bytes; F3 - 4 bytes; F4 - 9 bytes; F5 - 10 bytes; F6, and F7 - between 4 to 256 bytes). All ASCII serial numbers are stored one byte at a time, where the first ASCII character occupies the lowest byte address (first address).

Display Format: YYYY-1234 - size in FLASH - 8 bytes

```
SN 20030123 will be encoded as:
YYYY - 2003     ->   0x32 0x30 0x30 0x33  (ASCII)
1234 - 0123     ->   0x30 0x31 0x32 0x33  (ASCII)

<--------Addr-------->
00-01-02-03-04-05-06-07

32 30 30 33 30 31 32 33 ->  ASCII
```

Display Format: YYMM-1234 - size in FLASH - 8 bytes

```
SN 03030123 will be encoded as:
YYMM - 0303     ->   0x30 0x33 0x30 0x33  (ASCII)
1234 - 0123     ->   0x30 0x31 0x32 0x33  (ASCII)

<--------Addr-------->
00-01-02-03-04-05-06-07

30 33 30 33 30 31 32 33 ->  ASCII
```

Display Format: YYMMDD-1234 - size in FLASH - 10 bytes

```
SN 0405110123 will be encoded as:
YYMMDD - 040511 ->   0x30 0x34 0x30 0x35 0x31 0x31  (ASCII)
1234   - 0123   ->   0x30 0x31 0x32 0x33            (ASCII)

<-----------Addr----------->
00-01-02-03-04-05-06-07-08-09

30 34 30 35 31 31 30 31 32 33 ->  ASCII
```

Display Format: YYDDD-1234 - size in FLASH - 9 bytes

```
SN 042140123 will be encoded as:
YYDDD - 04214   ->   0x30 0x34 0x32 0x31 0x34  (ASCII)
1234  - 0123    ->   0x30 0x31 0x32 0x33       (ASCII)


<----------Addr--------->
00-01-02-03-04-05-06-07-08

30 34 32 31 34 30 31 32 33 -> ASCII
```

Display Format: 12345678 - size in FLASH - 8 bytes

```
SN 12345678 will be encoded as:
12345678 - 12345678  ->  0x31 0x32 0x33 0x34 0x35 0x36 0x37 0x38  (ASCII)


<-------Addr--------->
00-01-02-03-04-05-06-07

31 32 33 34 35 36 37 38    -> ASCII
```

Display Format: 1234 - size in FLASH - 4 bytes

```
SN 1234 will be encoded as:
1234 - 1234    ->   0x31 0x32 0x33 0x34  (ASCII)


<--Addr--->
00-01-02-03

31 32 33 34    -> ASCII
```

Display Format: Custom - size in FLASH - between 4 to 256 bytes

A SN entered using the "Custom" format is entered as a series of either an even number of HEX digits or any number of ASCII characters. This subsection deals only with storing the SN entered as a series of ASCII characters. The SN can be composed of anywhere between 4 and 256 characters.

```
SN sequence entered as ''02WX24S234'' will be encoded as:
0x30 0x32 0x57 0x58 0x32 0x34 0x53 0x32 0x33 0x34

<-----------Addr----------->
00-01-02-03-04-05-06-07-08-09

30 32 57 58 32 34 53 32 33 34 -> ASCII
```

Display Format: from File - size in FLASH - between 4 to 256 bytes

A SN entered using the "from File" format is read from the SN input file and can be a series of either an even number of HEX digits or any number of ASCII characters. Section 8.9 describes how to write a SN input file. This subsection deals only with storing the SN entered as a series of ASCII characters. The SN can be composed of anywhere between 4 and 256 characters.

```
SN sequence entered as ''O2WX24S234'' will be encoded as:
0x30 0x32 0x57 0x58 0x32 0x34 0x53 0x32 0x33 0x34

<-----------Addr----------->
00-01-02-03-04-05-06-07-08-09

30 32 57 58 32 34 53 32 33 34 ->  ASCII
```

8.4   Serial Number Output File

The **Serial Number Output File** field specifies the file where serialization information will be
saved after successful programming. The SN file contains the following data separated by tabs:

1. Serial Number Format (F0-6),

2. Serial Number,

3. SN action type (New SN[.], unmodified SN[u], overwritten SN[o], manual SN[m]),

4. Time and date when the SN was created,

5. Code File Name,

6. Model/Group/Revision text.

Below is an example of a SN file containing information on three consecutively created serial
numbers.

```
F0  200300011   m   ( Sat, Mar 29,2003, 10:09 )   AS010X02-1v2.txt     -01 R.0003-04-17
F0  200300012   .   ( Sat, Mar 29,2003, 10:43 )   AS010X02-1v2.txt     -01 R.0003-04-17
F0  200300013   u   ( Sat, Mar 29,2003, 10:43 )   AS010X02-1v2.txt     -01 R.0003-04-17
```

SNs can be saved as a unique SN per target device type, or saved as a unique SN across all devices
type. When a unique SN per target device type is saved then a separate SN file should be used
for each device type. If a unique SN across all device types is saved, then only one SN file name
should be used.

8.5   Model, Group, Revision

The software also allows the user to program each device with separate HEX or ASCII data. This
information is saved at a different memory location from the SN, and does not change after each
programming cycle. This is used to program static information that should be the same for each
target device, like the software revision number, model, etc. This extra data can be composed of
anywhere between 2 to 32 characters.

   This feature is enabled when the **ENABLE** check-box in the **Model/Group/Revision**
group is selected (see Figure 8.2). When enabled, specify the number of bytes to be reserved for
the desired text or data. Also, specify the memory address where this information will be stored
on the target device (must be even address). Finally, enter the text or data to be saved and the
format it should be stored in.

8.6   BarCode Scanner Setup

The BarCode Scanner can be used as an input mechanism for new SNs. When enabled, scanning a barcode will set the **Next SN** shown in Figure 4.1 to the value read by the BarCode Scanner. For convenience, the software can be configured to start the **Auto Program** procedure immediately after the BarCode Scanner reads a new number, by selecting the **Start AUTOPROGRAM following BarCode scan** check-box.

The BarCode Scanner should be connected to the programming PC computer in series with the keyboard using the Y cable or to the USB port. Refer to your BarCode Scanner manual for setup details.

Bar Code Scanner is enabled when the **ENABLE** check-box in the **BarCode Scanner** group is selected. Depending on the output from the scanner choose the right terminating character, although commonly the default CR (ENTER) character is used. Finally, change the **Display Format** to "Custom" and select the proper **Memory Format** as either HEX or ASCII.

Note

Only ASCII characters from 0x21 to 0xFE are accepted from the BarCode Scanner. Others characters like white characters (space, tab) are ignored. All characters are converted to lower case characters.

8.7   Device Serialization box

The **Device Serialization** box, located on the main programming dialog screen shown in Figure 4.1 and isolated in Figure 8.4), contains SN and model information. The first two read-only lines contain information taken from the target device during the read process. The first line shows the Model/Group/Revision text and the second line shows the SN. The following two lines contain SN and model information that will be programmed next generated automatically according to the Serialization setup.

The current target's SN and model information can be read at any time by pressing the **READ SN** button located in the **Device Serialization** group shown in Figure 8.4.

Figure 8.4: Seen in the main GUI window, this serialization status box will display the current device's serial number and the next serial number to be programmed.

8.8 Serialization Report Dialogue Screen

The Serialization Report Dialog Screen shows the results of the serialization procedure. The report contains detailed information on two SNs, specifically the highest SN and the one selected. The report also shows all programmed units along with their SN, and unmodified or overwritten SNs coming from reprogrammed units (New SN[.], unmodified SN[u], overwritten SN[o], manual SN[m]). Detailed information about all programmed units can be viewed using the Notepad text editor by pressing the **Notepad** button. A summary of created serial numbers is displayed in the white report box (see Figure 8.5). If for any reason a SN entry is wrong it can be removed from the database by pressing the **Delete SN** button (the delete operation is not reversible).
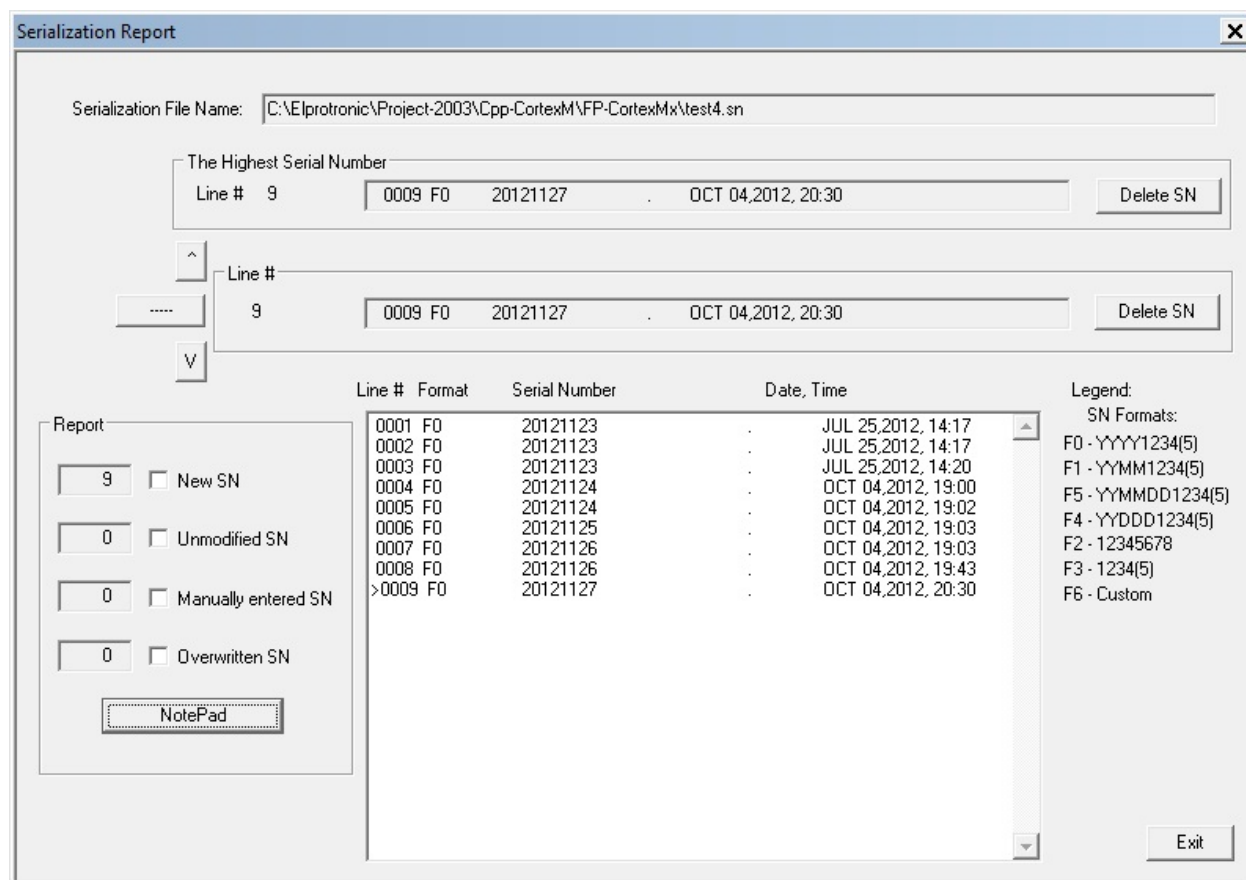
Figure 8.5: Serialization report gives a summary of all the serial numbers programmed using the selected file.

8.9   Serial Number Input File

The FlashPro-LM Programmer allows input of serial numbers from user created data files. When the data file is used change the **Display Format** to "from File" and select the proper **Memory Format** as either HEX or ASCII.

The SN data file should contain a proper header followed by a list of SNs specified using the following commands (a command is preceded by the # character). Any data entered after a semicolon (;) will be ignored and can be used for commenting. Use the following list of commands:

- **#SN_LIST** ;. Header, data file contains a SNs list.

- **#SN_SIZE** number; optional. Overwrite size of the custom defined serial number size (see Figure 8.2). If the #SN_SIZE is not specified, then the data specified in the serialization dialog screen is used.

- **#SN_HEX_MSB** ;optional. Select the HEX MSB first format regardless setup in the serialization dialog screen.

- **#SN_HEX_LSB** ;optional. Select the HEX LSB first format regardless setup in the serialization dialog screen.

  When the format is specified as the HEX format (#SN_HEX_MSB or #SN_HEX_LSB) then string can contain gaps between numbers or 0x on the front. All gaps will be removed. Up to 256 characters (gaps excluded) can be specified for one number. All characters must be located in one line for the same SN. Example:
  The HEX number:

  `A434BC5696AD10ACF0`

  can be specified as:

  `A434BC5696AD10AC`

  or

  `0xA434BC5696AD10AC`

  or

  `A434 BC56 96AD 10AC`

  or

  `A4 34 BC 56 96 AD 10 AC F0`

- **#SN_ASCII** ; optional. Select an ASCII format regardless setup in the serialization dialog screen.

- **#SN_PREFIX** string; optional.

- **#SN_SUFFIX** string; optional. Serial number can contains up to 32 characters. If part of characters are the same in specified serial number list, then the repeatable part can be specified in the #SN_PREFIX, or #SN_SUFFIX, and only modified part of serial numbers can be listed. Serial number is combined as a string starting from prefix, modified part and ending with suffix. For example if the following serial number should be created:

```
AB2007X-0001-BMR
AB2007X-0002-BMR
AB2007X-0003-BMR
```

can the SN be specified as follows:

```
#SN_PREFIX    AB2007X-
#SN_SUFFIX    -BMR
```

and list of following serial numbers:

```
0001
0002
0003
```

Prefix and /or suffix numbers can be modified in the list if required, e.g.

```
#SN_PREFIX    AB2007X-
#SN_SUFFIX    -BMR
0001
0002
0003
#SN_PREFIX   AB2007V-
0001
0002
0003
```

that defined following serial numbers:

```
AB2007X-0001-BMR
AB2007X-0002-BMR
AB2007X-0003-BMR
AB2007V-0001-BMR
AB2007V-0002-BMR
AB2007V-0003-BMR
```

Example of the Serial Number list (5 lines only in this example):

```
; ===================================================
;    Serial Number List
; SN format - ASCII
; ===================================================
#IEEE_SN_LIST
#SN_SIZE 12

WX5E2007001P
WX5E2007002P
WX5E2007003P
WX5E2007004P
WX5E2007005P
; ===================================================
```

The same Serial Number list with specified prefix /suffix:

```
; ===================================================
;   Serial Number List
; SN format - ASCII
; ===================================================
#IEEE_SN_LIST
#SN_SIZE 12
#SN_PREFIX       WX5E2007 ;any ASCII character
#SN_SUFFIX       P

001
002
003
004
005
; ===================================================
```

When the SN data file is prepared, then at the first the data base file should be opened (see Figure 8.2). When the desired Serial Number Format is selected, then using the "Serial Numbers File" button located in the main dialog screen (Figure 4.1) the desired SN file should be opened. Selected file is converted to final format and all listed serial numbers are verified with the data base file if there was note used before. If the specified SN have been used before, then these numbers are removed from the SN list. When the SN file is read and verified, then the pending SN list is displayed in the screen (Figure 8.6) with following information displayed on the top of the list:

- number of the SN found in data base and removed from the pending list,

- number of the Serial Numbers with incorrect size and removed from the pending list.

- number of the accepted SN.

When the "Paste to Notepad" button is pressed, then the pending Serial Number list can be saved in format ready to be used as a valid SN data file if required.

```
Serial numbers list from the file                                    ×

#SN_LIST
#SN_SIZE 6
;===================================================
;   0 - Number of the SN found in the data base
;        (removed from the list).
;   0 - Number of the SN numbers with incorrect size
;        (current setup - 6 char)(removed from the list)
; 10 - Number of the SN attached to the list.
;===================================================
001:     123456
002:     2A2211
003:     2AC212
004:     2AH213
005:     2DD214
006:     2YY215
007:     2ZZ216
008:     2NN217
009:     2MM218
010:     2QE219




        Paste to Notepad                    Exit
```

Figure 8.6: Serial numbers read from a file.

Chapter 9

Check Sum Options

Programming software has two groups of check sum (CS) calculation. The first group is used for internal programming verification and the second group can be used for firmware verification in application software.

The CS used for internal verification is calculating CS only for specified words in the code file regardless of the flash memory size, location etc. This CS is useful only inside the programmer, because programmer has all information about programmed and empty bytes location. This method is also useful if only part of the code is programmed in the flash (append option). All not programmed words in the programming process are ignored, even if these words are not empty in the flash.

The check sum used for internal programming verification is displayed in the Check Sum Group (Figure 9.1) (see the Main Dialog screen - Figure 4.1).

In the source line is displayed the arithmetic sum of the code contents with added contents of the serialization, model etc. if selected. Arithmetic sum is calculated as the sum of 16-bits unsigned words - result is 32 bits unsigned. Only programmed words are taken for calculation. All other not used words are ignored. All bytes are converted to 16-bits words as follows (for simplicity - format casting is not present in this example):

$$word = data[address] + (data[address + 1] << 8)$$

where address is even and incremented by 2.
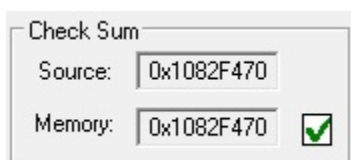
Figure 9.1: Check Sum information compared between the file and MCU memory.

67

In the memory line is displayed the CS result taken from the flash memory, calculated in the same way as the CS taken from the source. Only words defined in the source are taken from the flash memory for calculation.

Second group of the CS is custom defined Check Sum that can be used by firmware for code verification in the flash. Up to four CS block can be specified and CS results can be saved in the flash for verification. Size of each CS block and CS result location in flash are defined by the user.

The Check Sum Options dialog (Figure 9.2) is selected from following pull down menu:

$$Setup \rightarrow CheckSumOptions$$
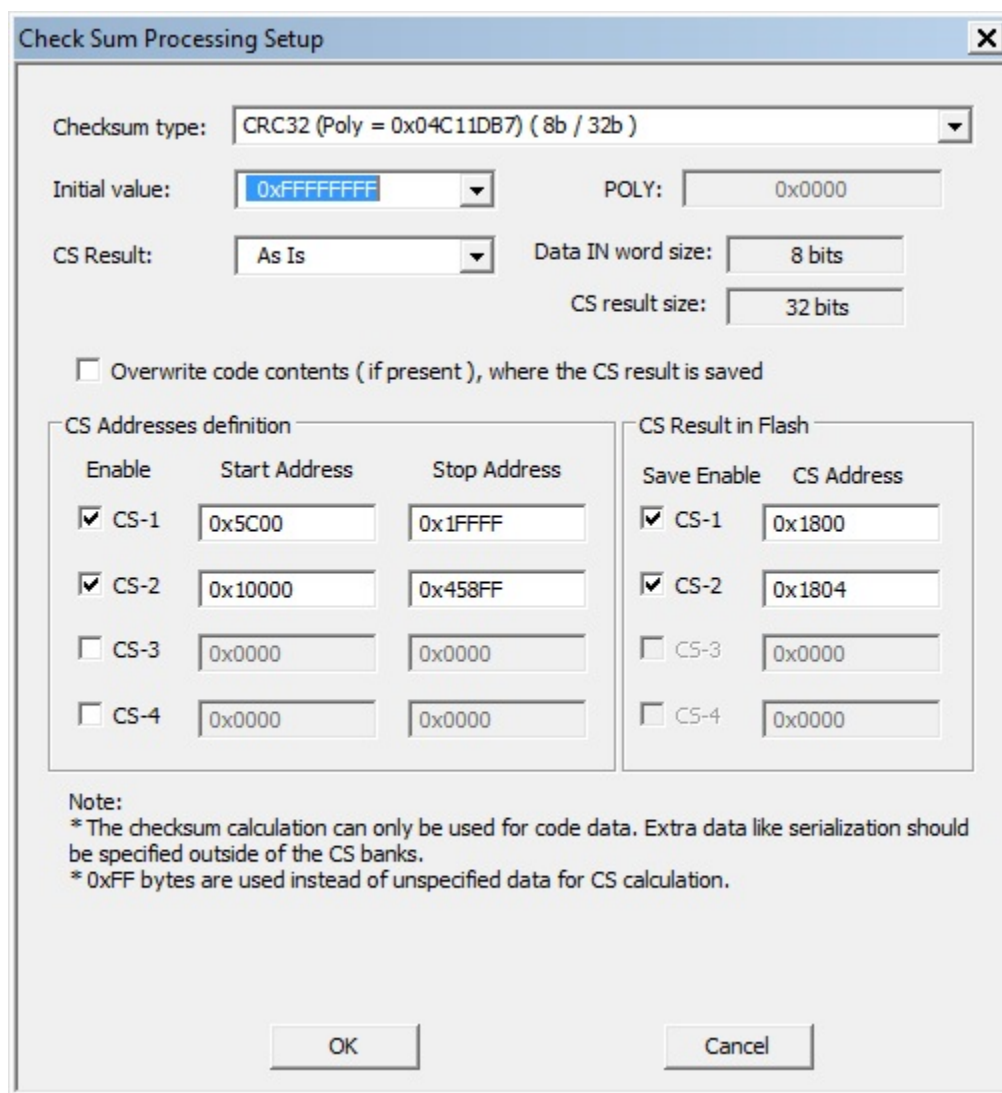
Start Address should be even, and the Stop Address should be odd. CS result address in the flash should be even. Make sure that the CS result is saved out of the CS block space. Otherwise the CS result will modify the contents of the CS inside the specified block. CS result after the second calculation would not be the same and CS result would be useless.

When the CS Result Save option is not selected then the CS of the selected block is calculated and CS result displayed in the report window only (Figure 9.3). This option can be used for CS code verification defined as the code form Start to End Addresses with 0xFF data in the not specified code location.

Type of the CS can be selected from the following list (Figure 9.4) Initial value for CS calculation can be selected as zero, all 0xFFs or as the Start Address from pull down menu (Figure 9.5).

CS result can be used As Is or can be inverted (Figure 9.6).

Data size (byte or 16 bits word) used for calculation and CS result size is displayed in the dialog screen as Data IN word size and CS Result size (Figure 9.2). Polynomial contents (if required) can be specified in the POLY edit line in HEX format (eg. 0x1234).

Figure 9.2: Check Sum Options.

Figure 9.3: Check Sum Report.
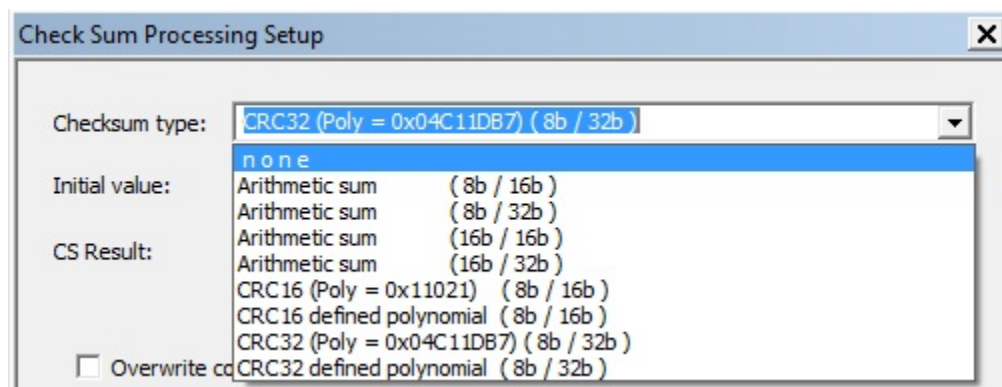
Figure 9.4: Check Sum Type.
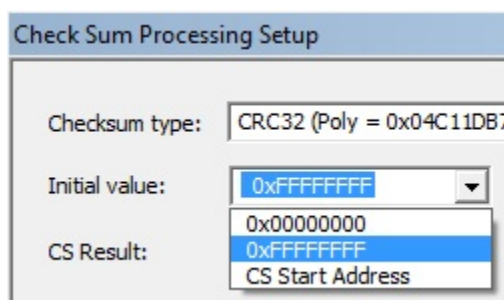
Figure 9.5: Check Sum Initial Value.
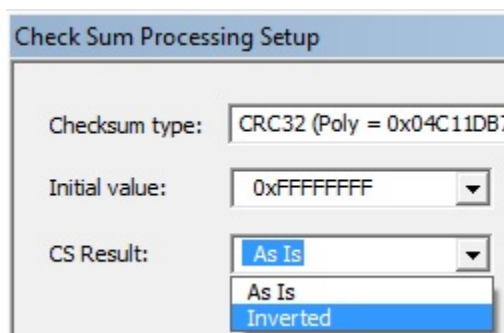
Figure 9.6: Check Sum Inverted Option.

9.1 Check Sum types

Following Check Sum types are implemented (Figure 10.4)

9.1.1 Arithmetic Sum ($\frac{8b}{16b}$)

Check Sum is calculated as modulo 16-bits sum of all bytes (unsigned) from Start to the End Addresses as follows

```
CS = CS_initial_value;
for ( addr = StartAddress;  addr <= EndAddress; addr++ )
{
    CS = CS + (unsigned int) data[addr];
}
CS = 0xFFFF & CS;
if( cs_inverted )
    CS = 0xFFFF ^ CS;
```

9.1.2 Arithmetic Sum ($\frac{8b}{32b}$)

Check Sum is calculated as modulo 32-bits sum of all bytes (unsigned) from Start to the End Addresses as follows

```
CS = CS_initial_value;
for ( addr = StartAddress;  addr <= EndAddress; addr++ )
{
    CS = CS + (unsigned long) data[addr];
}
CS = 0xFFFFFFFF & CS;
if( cs_inverted )
    CS = 0xFFFFFFFF ^ CS;
```

9.1.3 Arithmetic Sum ($\frac{16b}{16b}$)

Check Sum is calculated as modulo 16-bits sum of all 2-byte words (unsigned) from Start to the End Addresses as follows

```
CS = CS_initial_value;
for ( addr = StartAddress;  addr <= EndAddress; addr=addr+2 )
{
    CS = CS + (unsigned int)data[addr] + (unsigned int)data[addr+1];
}
CS = 0xFFFF & CS;
if( cs_inverted )
    CS = 0xFFFF ^ CS;
```

9.1.4 Arithmetic Sum ($\frac{16b}{32b}$)

Check Sum is calculated as modulo 32-bits sum of all 2-byte words (unsigned) from Start to the End Addresses as follows

```
CS = CS_initial_value;
for ( addr = StartAddress;  addr <= EndAddress; addr=addr+2 )
{
    CS = CS+(unsigned long)data[addr] + (unsigned long)data[addr+1];
}
CS = 0xFFFFFFFF & CS;
if( cs_inverted )
    CS = 0xFFFFFFFF ^ CS;
```

9.1.5   CRC16 (Poly 0x11201) - ($\frac{8b}{16b}$) (Named as CRCCCITT) and CRC16 defined polynomial - ($\frac{8b}{16b}$)

Check Sum is calculated as CRC16 from each bytes from Start to the End Addresses as follows

```
CS = CS_initial_value;
for ( addr = StartAddress;  addr <= EndAddress; addr++ )
{
    CS = CS_CRC16_8to16( (long)data[addr], CS );
}
CS = 0xFFFF & CS;
if( cs_inverted )
    CS = 0xFFFF ^ CS;
```

where

```
unsigned long CS_CRC16_8to16( long data, unsigned long crc )
{
    unsigned long tmp;
        tmp = 0xFF & ((crc >> 8) ^ data );
crc = (crc << 8) ^ crc_tab32[tmp];
    return( 0xFFFF & crc );
}
```

The CRC table is generated first as follows:

```
CS_init_crc16_tab( 0x1021 );  for CRC CCITT
CS_init_crc16_tab( CRC_def_POLY  );  for CRC16 defined polynomial
```

where

```
void CS_init_crc16_tab( unsigned short poly )
{
    int i, j;
    unsigned short crc, c;

    for (i=0; i<256; i++)
    {
        crc = 0;
        c   = ((unsigned short) i) << 8;

        for (j=0; j<8; j++)
{
            if ( (crc ^ c) & 0x8000 )
                crc = ( crc << 1 ) ^ poly;
            else
                crc =   crc << 1;
            c = c << 1;
        }
        crc_tab32[i] = (unsigned long)(0xFFFF & crc);
    }
}
```

9.1.6   CRC32 (Poly 0x04C11DB7) - $(\frac{8b}{32b})$ (Named as IEEE 802-3) and CRC32 defined polynomial - $(\frac{8b}{32b})$

Check Sum is calculated as CRC32 from each bytes from Start to the End Addresses as follows

```
CS = CS_initial_value;
for ( addr = StartAddress;  addr <= EndAddress; addr++ )
{
    CS = CS_CRC32_8to32( (long)data[addr], CS );
}
CS = 0xFFFFFFFF & CS;
if( cs_inverted )
    CS = 0xFFFFFFFF ^ CS;
```

where

```
unsigned long CS_CRC32_8to32( long data, unsigned long crc )
{
    return( ((crc >> 8) & 0x00FFFFFF) ^ crc_tab32[0xFF & (crc ^ data )] );
}
```

The CRC table is generated first as follows:

```
CS_init_crc32_tab( 0x04C11DB7 ) for IEEE 802-3
```

a polynomial of

$$x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$$

and

```
CS_init_crc32_tab( CRC_def_POLY ) for CRC32 defined polynomial
```

where

```
void CS_init_crc32_tab( unsigned long poly_in )
{
    int n, k;
    unsigned long c, poly;

    poly = 0L;
    for (n = 0; n < 32; n++)
    {
        poly <<= 1;
        poly |= 1L & poly_in;
        poly_in >>= 1;
    }

    for (n = 0; n < 256; n++)
    {
        c = (unsigned long )n;
        for (k = 0; k < 8; k++)
            c = c & 1 ? poly ^ (c >> 1) : c >> 1;
        crc_tab32[n] = c;
    }
}
```

Chapter 10

Script File

Programming sequence can be customized when is using a script file. Script file prepared as a text file (using any editor like notepad) can contains customized programming sequences in any order. Generally, all buttons available on the main dialogue screen can be used in the script file. All other options available on others screens like memory options, serialization type etc. can not be modified from the script file directly, but can be reloaded in fully using configuration file. From the script file any configuration files can be called at any time that allows to modify programmer configuration. This method can simplify programming process using script file and allows to use full options available in the programmer. Programming sequence conditions can be taken from user defined procedures attached as an independent DLL if required.

Programmer has two entry for taking the sequence from the script file:

1. by pressing the Script File button in the Main dialog, and

2. by using the -rf with the executable file.

10.1   Script button

The "Script" button is the dynamically programmable device action button that allows to take a desired action taken from the script file. The Script button has a name Script File - none (Figure 10.1) if the script file is not defined or Script: with used file name when the script file is active (Figure 10.2). When the Script button is pressed and the current script file is not active, then the Open File dialog is displayed and the desired script file should be selected. When the Script file button is not empty and the new script file if required, then the new file can be selected from the pull down menu - File→Open Script File.

The Script button is very useful for implementing a short programming sequence not present directly in the Device Action group buttons Below is an easy script file used for downloading two independent codes to target device - first code used for hardware test if possible, and when hardware is ok, then the second code is downloaded as the final code to target device. The same sequence

74
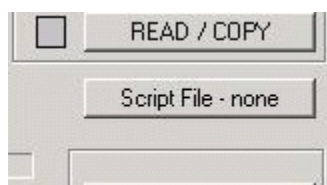
Figure 10.1: Script file not defined.
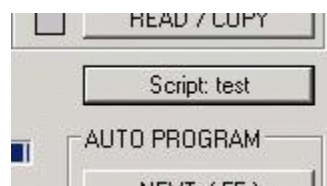
Figure 10.2: Script file active.

can be used with other buttons, but sequence should be always repeated, that of course is not convenient.

Using the notepad editor create the script file and save it eg. as the file "test.sf" or any other file name. See this chapter below for all available instructions that can be used in the script file.

```
;----------------------------------------------------------------------
; easy script file;
;----------------------------------------------------------------------

LOADCFGFILE C:\Program Files\Elprotronic\USB FlashPro-CC\test.cfg
LOADCODEFILE C:\Program Files\Elprotronic\USB FlashPro-CC\test.cfg
AUTOPROGRAM

; now the hardware is tested according to downloaded firmware
MESSAGEBOX    YESNO
  "Press YES when the test finished successfully."
  "Press NO  when the test failed."

IF BUTTONNO GOTO finish

LOADCFGFILE C:\Program Files\Elprotronic\USB FlashPro-CC\final.cfg
LOADCODEFILE C:\Program Files\Elprotronic\USB FlashPro-CC\final.cfg
AUTOPROGRAM

>finish
END
;----------------------------------------------------------------------
```

When the script file above is used then the first configuration file and the fist code file is downloaded and Autoprogram function is executed. When finished then the MCU firmware started (make sure that the first configuration allows to start the code when the Autoprogram is finished). Final code is downloaded when the test has been finished successfully.

Before running the script file the configuration files named test.cfg and final.cfg required in the setup should be created using the GUI software first. To do that connect target devices to programming adapter, select desired configuration and save the configuration file as test.cfg and create final configuration file in similar way.

10.2   Script file option

Programming sequence can be customized when using the -rf with the executable file (described in the "Setup and Configuration Load/Save" chapter) .

When the executable file FlashPro-LM.exe is called with a script path as an argument e.g.

$$FlashPro - LM.exe - rfC : \backslash ProgramFiles \backslash Elprotronic \backslash USBFlashPro - LM \backslash script.txt$$

or when the icon with the FlashPro-LM.exe and script file path is executed then programmer starts automatically programming sequences according to procedure specified in the script file. Access to other buttons are blocked. When script file sequence is finished then program is terminated. There is not option to modify the running sequence when script sequence is used. This option is useful in production, because nobody can modify sequence that has been prepared for the production purpose.

10.2.1   Script Limitations

- Up to a total of 1000 command lines can be used. Empty lines and comments are ignored.

- The stack supports a call depth of up to 50 CALLs (CALL inside CALL inside CALL...).

10.2.2   Command Syntax

- White spaces before instructions, labels, and comments are ignored.

- ; - start of a comment. All characters in the same line after the start of a comment are ignored.

A comment cannot be placed after a filename.

For example, when specifying a config file to be loaded, a path to a file must be given. This filename cannot be followed by a comment.

- > - start of a label. Place the label name after the character with no spaces in between.

A line with a label cannot also contain a command or another label.

For example, this would be illegal >START VCCOFF

10.2.3   Instructions

MESSAGE - message declaration. Contents must be placed between quotes below message declaration. Maximum of 50 content lines. Example:

```
MESSAGE
''Hello.''
''This is my script.''
```

GUIMSGBOX setting - Enable or disable pop-up message boxes in the GUI (warning, errors, etc.). Setting can be either ENABLE or DISABLE.

IFGUIMSGBOXPRESS option - Apply the option when a message box created by GUI is generated. Option can be OK or CANCEL.

MESSAGEBOX type - Create a pop-up message box with buttons. Contents must be placed between quotes below message declaration. Maximum of 50 content lines. Message box types are:

- **OK** - One button, OK.
- **OKCANCEL** - Two buttons, OK and CANCEL.
- **YESNO** - Two buttons, YES and NO.
- **YESNOCANCEL** - Three buttons, YES, NO, and CANCEL.

Example:

```
MESSAGE YESNOCANCEL
''You have three choices:''
''Press yes, no, or cancel.''
```

GOTO label - Jump to instruction immediately following the label.

CALL label - Call procedure starting at the instruction immediately following the label. Stack saves return address.

RETURN - Return from CALL.

IF condition operation - Test condition and if true then perform operation. The condition can be one of the following:

- **BUTTONOK** - OK button is pressed in the message box.
- **BUTTONYES** - YES button is pressed in the message box.
- **BUTTONNO** - NO button is pressed in the message box.
- **BUTTONCANCEL** - CANCEL button is pressed in the message box.
- **DONE** - Previous process, e.g. GO, Read File, etc., finished successfully.

  – **FAILED** - Previous process, e.g. GO, Read File, etc., failed.

The operation can be one of the following:

  – **GOTO** label
  – **CALL** label

SLEEP number - pause a number of milliseconds, between 1 and 100000.

F_LOADCFGFILE filename - Load configuration file. Provide a full path and filename.

F_LOADCODEFILE filename - Load code file. Provide a full path and filename.

F_APPENDCODEFILE filename - Append code file. Provide a full path and file name.

F_VCCOFF - Turn V_{cc} OFF from programming adapter to target device.

F_VCCON - Turn V_{cc} ON from programming adapter to target device.

<div align="center">

Vcc from FPA must be enabled first using configuration file.

</div>

F_VCCINMV - Set V_{cc} in mV, between 1800 to 3600 in steps of 100 mV.

F_RESET - Perform RESET function from main dialogue screen.

F_GO - Perform GO function from main dialogue screen.

F_ERASEFLASH - Perform ERASE FLASH function from main dialogue screen.

F_BLANKCHECK - Perform BLANK CHECK function from main dialogue screen.

F_WRITEFLASH - Perform WRITE FLASH function from main dialogue screen.

F_VERIFYFLASH - Perform VERIFY FLASH function from main dialogue screen.

F_BLOWFUSE - Perform BLOW FUSE function from main dialogue screen.

<div align="center">

Blows fuse regardless of enable option.

</div>

If the **BLOW FUSE** command is used then the security fuse will be blown even if the **Blow Security Fuse enable** option is disabled.

F_SETIMAGENUMBER number - Choose image number between 1 and 16 from FlashPro-LM Programmer internal memory.

F_STANDALONEMODE - Switch to **Standalone** mode.

F_INTERACTIVEMODE -Switch to **Interactive** mode.

TRACEOFF - Disable tracing.

TRACEON - Enable tracing and log to the **Trace-Scr.txt** file in the current working directory. This option is useful for debugging. The trace file will contain the sequence of all executed commands ran from the script file annotated with line numbers. Line numbers are counted without empty lines and without lines only containing comments.

END - End of script.

Below is an example of a script that executes the following sequence of commands:

1 Label START is created.

2 V_{cc} from programmer to target device is turned OFF.

3 Message box notifies the user of V_cc setting and asks for permission to proceed with buttons OK and CANCEL. The program will halt here until a button is pressed.

9 If CANCEL was pressed then GOTO finish label (ends the script).

10 If CANCEL was not pressed (in this case this implies that OK was pressed) then load configuration file **test-A.g430cfg** to the FlashPro-LM Programmer . Configuration file **test-A.cfg** should be prepared before running this script using **Interactive** mode.

11 Message box asks the user to proceed. The program will halt until OK is pressed.

13 The FlashPro-LM Programmer will program the target device using the GO function.

14 Message box asks the user if the test succeeded giving a YES or NO choice.

17 If NO was pressed then GOTO START label (start of script).

18 If NO was not pressed (in this case this implies that YES was pressed) then load configuration file **finalcode.g430cfg** to the FlashPro-LM Programmer .

19 The FlashPro-LM Programmer will program the target device using the GO function. The new configuration changed the code file.

20 Script jumps to the beginning using GOTO START. This can be used to wait for the next target device to be connected.

21 Label finish is created.

22 Script ends.

```
;====================================================
; Script file - demo program
;----------------------------------------------------
>START
  VCCOFF
```

```
  MESSAGEBOX OKCANCEL
    "VCC if OFF now. Connect the test board."
    "When ready press the button:"
    " "
    "OK - to test the board"
    "CANCEL - to exit from program"

 IF BUTTONCANCEL GOTO finish
 F_LOADCFGFILE C:\Elprotronic\Project\Cpp-Net\GANG430U\test-A.G430cfg

 MESSAGEBOX OK
   "Press OK to download the test program."

 F_GO

 MESSAGEBOX YESNO
   "Press YES when the test finished successfully."
   "Press NO when the test failed."

 IF BUTTONNO GOTO START

 F_LOADCFGFILE C:\Elprotronic\Project\Cpp-Net\GANG430U\finalcode.G430cfg

 F_GO

 GOTO START

>finish
  END
;=========================================================
```

Chapter 11

Setup and Image Load and Save

Programming software can save configuration settings in the configuration files or save the whole setup configuration with used code contents and save it in the encrypted setup file . This allows the user to create several configuration or setup files, one for a particular task, and thus eliminates the need to manually change settings every time a different configuration is desired. Furthermore, the config.ini file contains the most recently used settings and those settings will be used as default whenever the software is started.

11.1   Load and Save Setup

To create a configuration file simply select Save Setup from the File menu. Current settings will be saved for future use. To restore configuration settings select Load Setup from File menu and select a file containing the settings you wish to restore.

In order to prevent accidental setup changes the FlashPro-LM Programmer provides the option to Lock configuration settings. When the user selects the Lock/Unlock Setup option from the Setup menu, the FlashPro-LM Programmer will prevent the user from modifying the setup. The only options that are available when the programmer is locked are Verify, Read, Autoprogram and Next. Notice that the Next button will immediately change to implement the Autoprogram function. To unlock the programmer the user must select the Lock/Unlock Setup option from the Setup menu.

11.2   Load and Save Image

The image contains more than the programmer's configuration only, but can also the code and the CSM password used in the image. Contents of the image file are encrypted, so it is not possible to read the contents of the code downloaded to the target device. When the image is opened then the same decryption key must be entered as was used in the encryption process, otherwise decryption will not succeed. Encryption key depends on the type of software used (e.g. FlashPro-LM Programmer), the chosen password or destination PC's hardware fingerprint. The image

file created with the FlashPro-LM Programmer software cannot be used with the FET-Pro430 or GangPro430. Image file is CRC protected and CRC check is performed when the file is loaded.

The image can be protected with the destination PC hardware fingerprint or password protected. This allows to create the image that can be used only on the specific PC when the image is encrypted with the destination PC hardware fingerprint (useful in production) or to create the image that can be used only when the correct password is entered every time when the image is open. When the image is locked then only major buttons like Autoprogram or Verify are active and only a few pull-down menu items are accessible. All options that allow the user to read the code contents are blocked.

When the new image is create then it is recommended to select the New Setup from pull down menu and set the default option of all parameters and names used in the programmer. Next, choose the desired processor, code file and programming options (see all available options described in this manual). When this is done, verify if programmer works as expected. Subsequently, then the current setup can be saved as an image file. Select the Save Image as.. from File pull down menu. Following dialog will be displayed (Figure 11.1) that allows to select desired image options.

The screen shown in Figure 11.1 allows the user configure what type of security will be used to protect the image file. Three options are available, however for all three options the contents of the code file are always encrypted and cannot be read.

1. **Any PC** - image can be opened on any computer using FlashPro-LM Programmer software. It can be used for programming only.

2. **Any PC - Password protected** - image can be opened on any computer using the FlashPro-LM Programmer software, but only after the desired password has been entered.

3. **Selected PC - Hardware Fingerprint number** - image can be opened only on the dedicated computer with the same hardware fingerprint number as the number entered in Figure 11.1 under "Selected PC - Hardware Fingerprint #". An example usage scenario would involve calling an intended user to provide the hardware fingerprint number of their computer and entering it within this configuration window. This will restrict opening this image to only the dedicated computer running FlashPro-LM Programmer software.

   PC's hardware fingerprint used for the image can be read by selecting the "PC Hardware fingerprint number" option from pull down menu

$$About/Help \rightarrow PC\ Hardware\ fingerprint\ number$$

   The message box show in Figure 11.2 is displayed when the option above is selected.

The screen shown in Figure 11.1 also allows the user to configure what type of locking options will be used for the image.
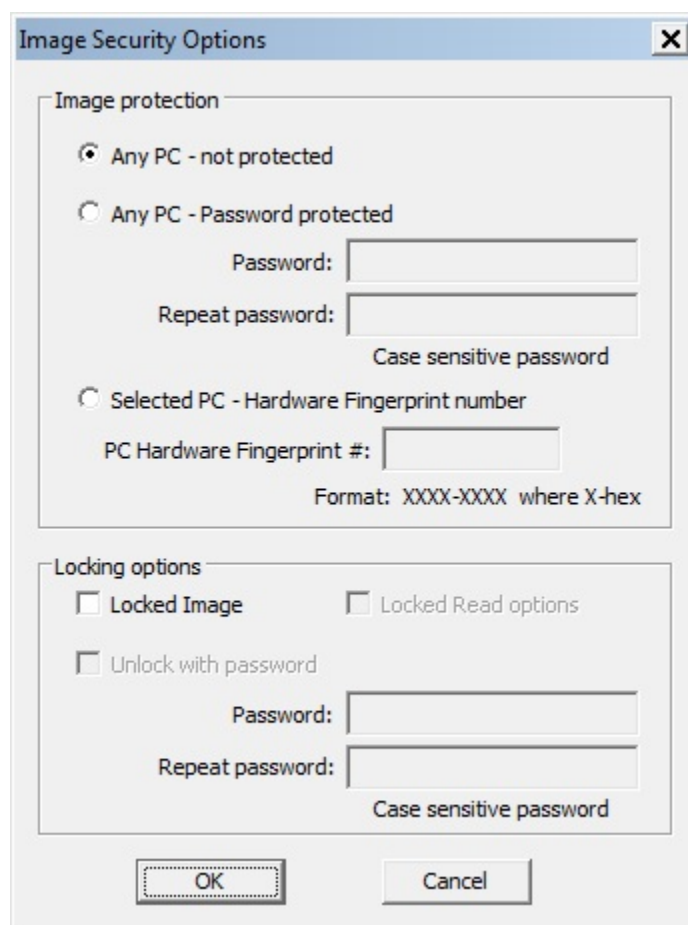
Locked Image

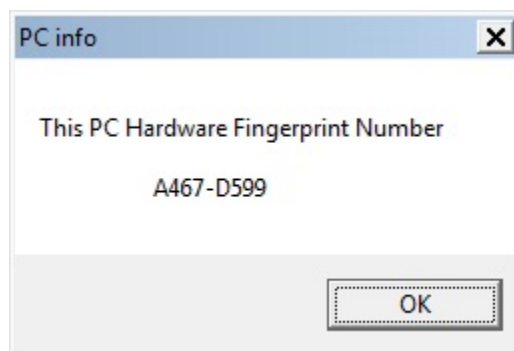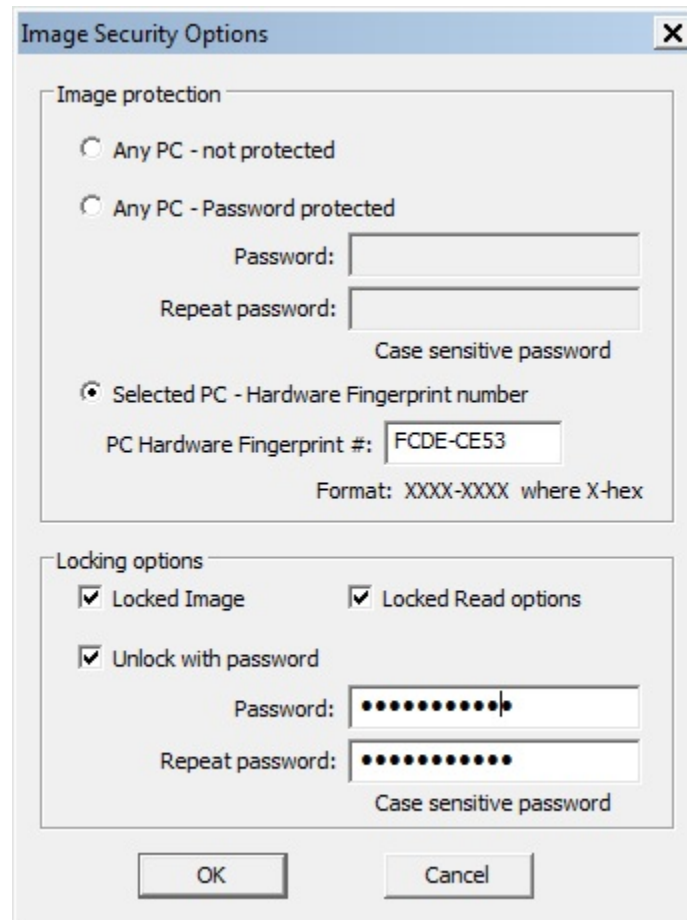**Figure 11.1:** Image Options.

Figure 11.2: PC Hardware fingerprint.

1. When not selected, then setup is not locked. All contents can be modified and all buttons are accessible.

2. When selected then setup is locked. Almost all buttons are disabled (grayed) and almost all items in the pull down menu are disabled. It is possible to select to permanently lock the image, or select an option to unlock the image with a password. The unlock password cannot be the same as the password used for opening the image.

Locked Read options   When selected then code viewers and READ button are blocked and not allowee to read code contents that are downloaded to target device. If the security fuse is blown after programming the target device, then code cannot be seen by the staff downloading code to target devices.

Unlock with password   When setup is locked then it is possible to select option "Unlock with password" and specify up to 16 characters unlocking password. Password is case sensitive. Shown in Figure 11.3 is a "Setup Security Options" dialogue screen with selected options Setup protected with PC's "hardware fingerprint" number, locked and unlocked with password.

By default, image is not protected and not locked. This allows to create unprotected image and open it at any time on any PC without restrictions. All buttons and items on the dialogue screen are enabled.

Figure 11.3: Image Options with security settings enabled and the password lock and unlock options specified.

11.3   Commands combined with the executable file

Image file or configuration setup file (or Code file) can be opened using Load Setup (Load Code / Password File) option from File menu or can also be opened using command line combined with the executable file name. Following command line switches are available:

-prj Image file name (Open Image file)

-sf Setup file name (Open Setup file)

-cf Code file name (Open Code file)

-nf IEEE/SN file name (Open IEEE addresses / Serial number list file)

-rf Script file name (Run programming sequence from the Script File)

-lock

Note:

When the -cf option is used, then code file name saved in the setup file (configuration file) is ignored and code file name specified with key -cf is used. When the -prj option is used, then the -sf, -cf, -rf options are ignored.

Using Windows START button (left bottom) select Run.. Using Browse.. find and select executable file.

$$C: \backslash Program\ Files \backslash Elprotronic \backslash USB\ FlashPro - LM \backslash FlashPro - LM.exe$$

and at the end enter the required key with the name of the setup file e.g.

$$C: \backslash Program\ Files \backslash Elprotronic \backslash USB\ FlashPro - LM \backslash FlashPro - LM.exe$$

$$-sf\ E: \backslash Elprotronic \backslash MFG \backslash prg - 04.cfg$$

To fully lock the setup the extra key "-lock" can be added in the command line e.g.

$$C: \backslash Program\ Files \backslash Elprotronic \backslash USB\ FlashPro - LM \backslash FlashPro - LM.exe$$

$$-lock\ -sf\ E: \backslash Elprotronic \backslash MFG \backslash prg - 04.cfg$$

The following setup can be created using Shortcut options which creates an icon located on the desktop. To do that move the cursor to inactive desktop area, click right mouse button and select New.

Using Browse.. in the Create Shortcut dialogue box select the following executable file

$$C: \backslash Program\ Files \backslash Elprotronic \backslash USB\ FlashPro - LM \backslash FlashPro - LM.exe$$

and at the and add the required command keys e.g.

$$C : \backslash Program\ Files \backslash Elprotronic \backslash USB\ FlashPro - LM \backslash FlashPro - LM.exe$$

$$-lock\ -sf\ E : \backslash ElproTronic \backslash MFG \backslash prg - 04.cfg$$

Click button Next and follow instructions to create icon. Clicking on the selected icon FlashPro-LM programming software will start with the selected configuration setup, and locked if specified.
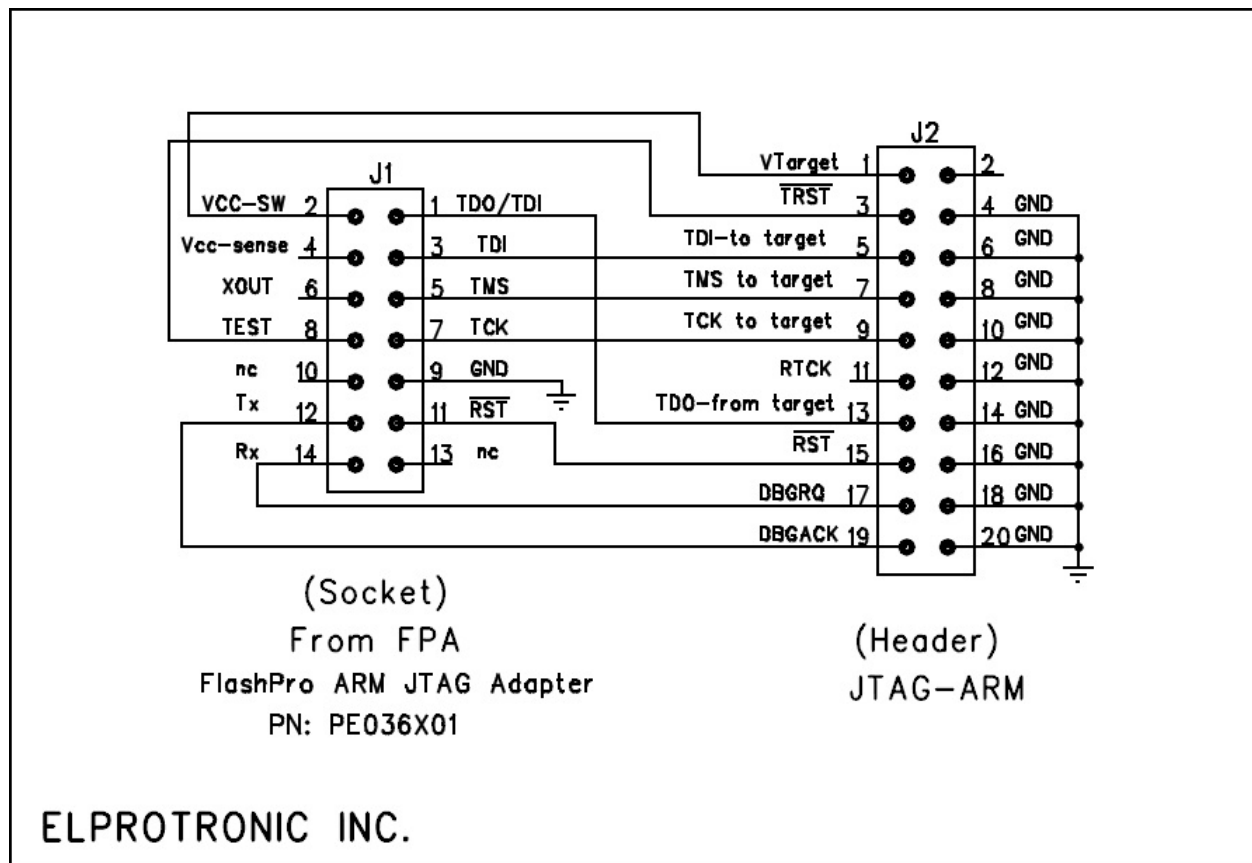
Chapter 12

Target Connection

Figure 12.1: Adapter.